Int. J. Appl. Math. Comput. Sci., 2011, Vol. 21, No. 1, 173–192
DOI: 10.2478/v10006-011-0013-2

HIERARCHICAL RESIDUE NUMBER SYSTEMS WITH SMALL MODULI AND SIMPLE CONVERTERS

TADEUSZ TOMCZAK

Institute of Computer Engineering, Control and Robotics
Wrocław University of Technology, Janiszewskiego 11/17, 50–372 Wrocław, Poland
e-mail: tadeusz.tomczak@pwr.wroc.pl

In this paper, a new class of Hierarchical Residue Number Systems (HRNSs) is proposed, where the numbers are represented as a set of residues modulo factors of $2^k \pm 1$ and modulo 2^k. The converters between the proposed HRNS and the positional binary number system can be built as 2-level structures using efficient circuits designed for the RNS $(2^k - 1, 2^k, 2^k + 1)$. This approach allows using many small moduli in arithmetic channels without large conversion overhead. The advantages resulting from the use of the proposed HRNS depend on the possibility of factorisation of moduli $2^k \pm 1$.

Keywords: digital arithmetic, digital circuits, residue number system, VLSI.

1. Introduction

In Residue Number Systems (RNSs), an integer X is represented as a set of *residues* X_i modulo given coprime *moduli* M_i (Soderstrand *et al*., 1986). The moduli set is called the system *base* and the residues are called *residue digits*. The *dynamic range* of the system, i.e., the number of possible number representations, is defined as the product of all RNS moduli. On the residue representation, addition, subtraction and multiplication can be done without carry propagation between residue digits.

Since residue digits are usually much smaller than the represented number X, arithmetic circuits for individual digits are smaller and faster than circuits for the binary representation of X. This property allows building arithmetic units for large numbers as a set of small and fast circuits (Mohan, 2002). The residue arithmetic also allows significant reduction of power consumption, especially in multipliers (Chokshi *et al*., 2009) and multipliers-accumulators (Piestrak and Berezowski, 2008a; 2008b). However, the implementation of the other arithmetic operations (e.g., division, sign detection, number comparison, overflow detection) is complex in the RNS and should be avoided. Additionally, the connection of residue arithmetic channels with the rest of a digital system has to be performed using converters between the RNS and the binary positional number system.

RNSs are especially useful in algorithms where multiplications and additions dominate, e.g., in digital signal and image processing (Wnuk, 2008), where RNS multiply-and-accumulate circuits can be used (Piestrak and Berezowski, 2008b). An RNS-based digital image processing application is proposed by Wang *et al*. (2004) while RNS-based Finite Impulse Response (FIR) filters are described by Conway and Nelson (2004) as well as Piestrak and Berezowski (2008b). An interesting review of RNS potential and applications is presented by Mohan (2002) and Soderstrand *et al*. (1986).

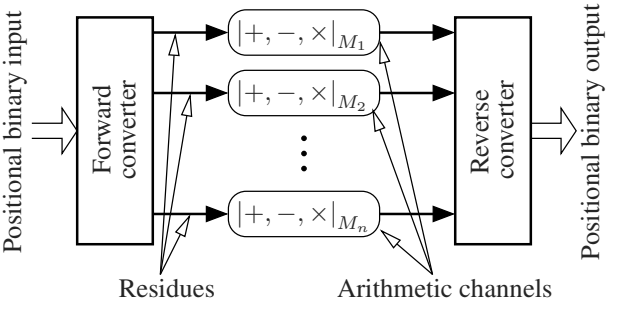


Fig. 1. General scheme of RNS processing.

The general scheme of an RNS circuit is shown in Fig. 1. The circuit consists of three main parts: the forward converter, the residue arithmetic channels and the reverse

converter. The forward converter translates binary input numbers into the set of residue digits. Next, for the residue representation, additions, subtractions and multiplications are executed using independent circuits performing calculations modulo appropriate moduli. The final part of the circuit is the reverse converter, which computes the positional binary representation from the residue representation obtained after the computations have been done using arithmetic channels modulo M_i. The converters introduce hardware overhead, thus the use of the RNS is justified only if savings in residue channels outweigh the conversion cost. It should also be noted that all results are computed modulo the RNS dynamic range and there is no simple method to detect overflow.

One of the main problems in RNSs is the selection of the system base. The RNS base should be chosen for each application individually to get a suitable dynamic range, circuit speed and complexity. For the required dynamic range, a trade-off between the speed of arithmetic units and conversion complexity has to be found. Small moduli allow building small and fast arithmetic units, but the number of moduli in the RNS base and, consequently, the complexity of converters grow. On the other hand, arithmetic operations for large moduli could be costly. The combination of small moduli, a large dynamic range and simple converters is possible in Hierarchical Residue Number Systems (HRNSs) .

In HRNSs, the values of all or only some residue digits are represented in residue systems with dynamic ranges smaller than the range of the main system (Akušskij and Judickij, 1968; Yassine, 1992; Skavantzos and Abdallah, 1999). The system used to represent residue digits is called the *lower level RNS*, whereas the system whose digits are represented in the lower level RNS is called the *higher level RNS*. The digits of the lower level RNS can also be represented in the next RNS leading to a multilevel HRNS. The highest system in the hierarchy is called the *top level RNS*, and the lowest system is called the *bottom level RNS*.

The dynamic ranges of lower level RNSs can be chosen in two ways. In the first approach (Akušskij and Judickij, 1968; Yassine, 1992), the dynamic ranges of the lower level RNS are large enough to represent intermediate results. For example, if multiplication is to be done, then the lower level RNS range has to be equal to at least the higher level modulus square. The advantage of this solution is that the same moduli can be used for the representation of different residue digits. As an example, consider the top level RNS $(17, 19, 20, 21)$ and single multiplication as an arithmetic operation to compute. Since the maximum values of the residue digits are $16, 18, 19$ and 20, the dynamic ranges for the bottom level RNS have to be at least $16^2 + 1 = 257, 18^2 + 1 = 325, 19^2 + 1 = 362$ and $20^2 + 1 = 401$. Thus, the RNS $(3, 4, 5, 7)$ with the dynamic range 420 can be used for all four digits. It is then possible to build an HRNS with the range $17 \cdot 19 \cdot 20 \cdot 21 > 2^{17}$ with 3-bit moduli. Unfortunately, the main disadvantage of this solution is fast growth of lower level RNS dynamic ranges. Additionally, the converters between consecutive levels have to be used after a small number of arithmetic operations—in the above example, after one multiplication.

In the second approach (Skavantzos and Abdallah, 1999), the top level RNS base is chosen from numbers factorisable into small factors. The lower level RNS bases are then built from factors for corresponding moduli. In this method, the range of the lower level RNS is equal to moduli from the higher level base. Thus, performing calculations in the lower level RNS is identical with performing them modulo higher level moduli. The conversion between consecutive levels can be done once for all arithmetic operations, which results in low hardware overhead. However, the main disadvantage of this idea is the difficulty with finding the top level RNS base. In the work of Skavantzos and Abdallah (1999), this base is chosen from moduli $2^{2k} - 1$ and the bottom level RNSs are $(2^k - 1, 2^k + 1)$. This allows implementing converters and arithmetic units as simple structures based on Eqns. (12) and (13) presented in the next section. However, since all the moduli in the RNS base have to be coprime, in the HRNS by Skavantzos and Abdallah (1999) the moduli width difference can be large. Moreover, some moduli values can be close to the system dynamic range, so that any advantages are lost.

In this paper, a new method for HRNS base construction is proposed. The base includes the moduli $2^k \pm 1$ factorisable into small divisors. This approach allows building input/output converters as two-level circuits, as shown in Fig. 2. In the top level RNS, the conversion between large numbers (close to the system range) and the residues modulo $2^k \pm 1$ is done. In the bottom level RNS, transformations between residues modulo $2^k \pm 1$ and residues modulo factors of $2^k \pm 1$ are performed. We shall show that, due to an efficient implementation of operations modulo $2^k \pm 1$ for large numbers, the area and critical path delay of the proposed two-level converters are small. Additionally, arithmetic operations are performed modulo small moduli, and thus adders and multipliers can be implemented as small and fast circuits.

This paper is organized as follows. Section 2 offers theoretical background including basic definitions, reverse conversion algorithms and methods of efficient implementation of residue operations using the periodicity property of the series of powers of 2 taken modulo M_i. Section 3 contains the proposition of a new HRNS class with the analysis of the multiplier's complexity and detailed formulas describing the conversion between the proposed HRNS and the positional binary number system. Section 4 presents detailed information about the implementation of reverse converters for the proposed HRNS and the com-

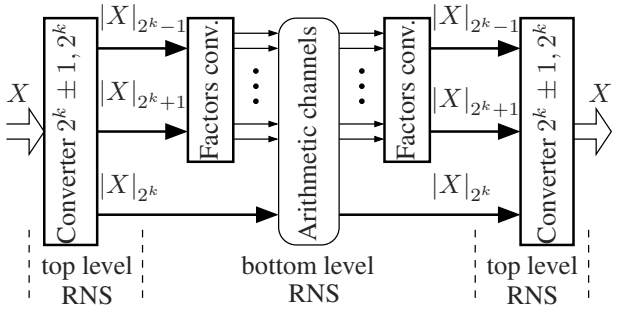


Fig. 2. Structure of the proposed HRNS.

parison of converters hardware complexity to other RNSs. Section 5 summarizes the paper and offers conclusions.

2. RNS basics

The RNS base is a set of n positive, co-prime moduli $M_i > 1$. The RNS dynamic range M is given as a moduli product

$$M = \prod_{i=1}^{n} M_i. \tag{1}$$

According to the Chinese remainder theorem (Soderstrand *et al.*, 1986), any integer $X \in [A, A+M)$ for any A can be uniquely represented in the RNS as an n–tuple $(X_1, X_2, \ldots, X_n)$. The residue digits $X_i \in [0, M_i)$ are remainders of the division of X by M_i. This is denoted by

$$X_i = |X|_{M_i} \Leftrightarrow X = \left\lfloor \frac{X}{M_i} \right\rfloor \cdot M_i + X_i. \tag{2}$$

Two integers X and Y which have the same residue when divided by a specific moduli M_i are called *congruent* modulo M_i. This is denoted by

$$X \equiv Y \mod M_i. \tag{3}$$

Addition, subtraction and multiplication can be executed for each modulus M_i independently,

$$X \odot Y = W \Leftrightarrow |X_i \odot Y_i|_{M_i} = W_i, \quad 1 \le i \le n, \tag{4}$$

where $X, Y, W \in [A, A+M)$, $\odot \in \{+, -, \cdot\}$ and $Y_i = |Y|_{M_i}$, $W_i = |W|_{M_i}$. The result W of operations defined by (4) is the computed modulo M, but there is no simple way to detect overflow. The dynamic range of the RNS can be extended by adding additional moduli to the base, but, unlike in weighted number systems, the values of additional digits cannot be easily obtained during computations. Accordingly, the dynamic range of the RNS must be large enough to represent even the largest result of computations.

2.1. Periodicity property. The periodicity property of the series of powers of 2 taken modulo M_i presented by Piestrak (1994) can be used for efficient implementation of residue arithmetic circuits and forward and reverse converters. The periodicity property results from Euler's theorem (Biernat, 2007), which states that for coprime, positive integers X and M,

$$\left|X^{\varphi(M)}\right|_M = 1, \tag{5}$$

where $\varphi(M)$ is the totient function. Thus, the residues for consecutive powers of X repeat with a period equal to at most $\varphi(M)$.

Let $2^j, 2^k, j < k$ denote two different powers of 2 and the *distance* between 2^j and 2^k be defined as $k - j$. According to Piestrak (1994), the *period* $P(M_i)$ of the odd modulus M_i is defined as the minimum distance between two different powers of 2, for which residues modulo M_i are equal, i.e.,

$$P(M_i) = \min(k-j), \text{ where } \left|2^j\right|_{M_i} = \left|2^k\right|_{M_i}. \tag{6}$$

The *half-period* $HP(M_i)$ of the odd modulus M_i is the minimum distance between two different powers of 2, for which residues modulo M_i are additive inverses, i.e.,

$$HP(M_i) = \min(k-j) \text{ where } \left|2^j\right|_{M_i} = \left|-1 \cdot 2^k\right|_{M_i}. \tag{7}$$

$P(M_i)$ exists for any odd M_i, whereas $HP(M_i)$ exists for some M_i only. If $HP(M_i)$ exists, then $P(M_i) = 2 \cdot HP(M_i)$. As an example, consider $M_i = 9$: $\left|2^0\right|_9 = 1, \left|2^1\right|_9 = 2, \left|2^2\right|_9 = 4, \left|2^3\right|_9 = 8 = -1, \left|2^4\right|_9 = 7 = -2, \left|2^5\right|_9 = 5 = -4, \left|2^6\right|_9 = 1, \ldots$. Thus, $P(9) = 6$ and $HP(9) = 3$.

The periodicity property is very often used for moduli $2^k \pm 1$, because $P(2^k - 1) = k$ and $HP(2^k + 1) = k$. A typical example is the forward converter computing residues modulo $2^k \pm 1$. Let a positive integer X be represented in the positional binary number system as an l-bit vector $x_{l-1} \ldots x_1 x_0$. The value of X is

$$X = \sum_{j=0}^{l-1} 2^j \cdot x_j. \tag{8}$$

Let

$$B_a = \left| \left\lfloor \frac{X}{2^{a \cdot k}} \right\rfloor \right|_{2^k} \tag{9}$$

denote the a-th k-bit field taken from a binary representation of X. Since

$$\left|2^k\right|_{2^k-1} = 1 \tag{10}$$

and

$$\left|2^k\right|_{2^k+1} = |-1|_{2^k+1}, \tag{11}$$

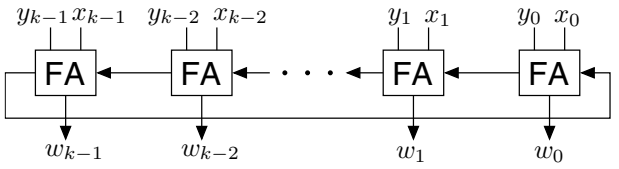


Fig. 3. Adder with end around carry performing operation $W = |X + Y|_{2^k-1}$. Here x_i, y_i and w_i denote the i-th bits of binary representations of X, Y and W.

the residues of a binary number X modulo $2^k \pm 1$ are defined as

$$\begin{aligned} \big|X\big|_{2^k-1} &= \Big| \sum_{a=0}^{\lfloor l/k \rfloor} \sum_{j=0}^{k-1} 2^j \cdot x_{a\cdot k+j} \Big|_{2^k-1} \\ &= \Big| \sum_{a=0}^{\lfloor l/k \rfloor} B_a \Big|_{2^k-1} \end{aligned} \tag{12}$$

and

$$\begin{aligned} \big|X\big|_{2^k+1} &= \Big| \sum_{a=0}^{\lfloor l/k \rfloor} (-1)^a \cdot \sum_{j=0}^{k-1} 2^j \cdot x_{a\cdot k+j} \Big|_{2^k+1} \\ &= \Big| \sum_{a=0,\ a \text{ even}}^{\lfloor l/k \rfloor} B_a - \sum_{a=1,\ a \text{ odd}}^{\lfloor l/k \rfloor} B_a \Big|_{2^k+1} \end{aligned} . \tag{13}$$

According to Eqns. (12) and (13), the residues $|X|_{2^k\pm1}$ can be computed as a sum or a difference modulo $2^k \pm 1$ of k–bit fields B_a. The addition can be done with multi-operand modulo adders (MOMAs) built using carry-save adders (CSAs) with end-around carry (EAC). Detailed design guidelines and analysis of a periodicity based MOMA for various moduli are presented by Piestrak (1994).

The idea of the use of EAC is shown in Fig. 3 on the basis of a ripple-carry adder (RCA) built from full-adder cells (FA). Since the residue modulo $2^k - 1$ for the k-th bit of the output sum is 1. Thus the k-th bit can be added in the least significant position of the adder. Notice that the adder shown in Fig. 3 can produce the result equal to $2^k - 1$ instead of 0, e.g., when adding $2^k - 2$ and 1. The adder is then called an adder with *double zero representation*. When the double zero representation is undesirable, additional circuits are required or other structures of adders modulo $2^k - 1$ should be used, e.g., parallel prefix adders (PPAs) (Biernat, 2007). The VHDL library of residue adders and multipliers using periodicity property is presented by Zimmermann (1998).

2.2. Reverse conversion. Reverse conversion algorithms are based on the Chinese Remainder Theorem (CRT) or Mixed Radix Conversion (MRC). In the classical CRT converters, the value of X is computed as

$$X = \left| \sum_{i=1}^{n} \frac{M}{M_i} \left| \cdot \left(\frac{M}{M_i} \right)^{-1} \right|_{M_i} \cdot X_i \right|_M . \tag{14}$$

In the MRC converters (Soderstrand *et al.*, 1986), X is defined as

$$X = \sum_{i=1}^{n} a_i \cdot \Big(\prod_{j=1}^{i-1} M_i\Big), \tag{15}$$

where

$$a_i = |r_i|_{M_i} \tag{16}$$

and

$$\begin{aligned} r_1 &= X, \\ r_i &= (r_{i-1} - a_{i-1}) \cdot \big|(M_{i-1})^{-1}\big|_{M_i} . \end{aligned} \tag{17}$$

Recently, an algorithm called' the new Chinese remainder theorem II, has been developed (Wang, 2000). For a system defined by two moduli (M_1, M_2), the reverse conversion can be performed according to the equation

$$X = X_2 + |q_{2,1} \cdot (X_1 - X_2)|_{M_1} \cdot M_2, \tag{18}$$

where $q_{2,1} = \big|M_2^{-1}\big|_{M_1}$ and $X \in [0, M_1 \cdot M_2)$. The only requirement for M_1 and M_2 is being *relatively prime*. Converters based on (18) can also be used in the RNS with the base consisting of any number of pairwise prime moduli. The converters are then built as multi-level structures (Fig. 4). On each level the moduli are grouped in pairs and for each pair Eqn. (18) is applied. For each pair the new residue modulo product of the moduli from the pair is computed. The computed residues are then the input to the next level, where they are grouped in new pairs and the whole process is repeated. On the last level two moduli roughly equal to $\sqrt{M}$ are used. Since in (18) only one modulo operation performed on the difference $X_1 - X_2$ is required and the other operations (multiplication by M_2 and addition of X_2) are done using positional arithmetic, this approach could bring meaningful simplification compared to the CRT, where a multi-operand addition modulo product of all moduli is needed.

Besides the above general algorithms, there are also known reverse converters optimized for specific moduli sets (Cao *et al.*, 2003; 2007; Molahosseini *et al.*, 2010; Wang *et al.* 2000; 2003;). Among many different moduli sets the set $(2^k - 1, 2^k, 2^k + 1)$ was often investigated and many efficient reverse converters are developed (Piestrak, 1995; Wang *et al.*, 2000; Mohan, 2001; Bi *et al.*, 2004).

3. New HRNS class

In the proposed HRNS, the top level base is chosen from moduli $2^k \pm 1$ factorisable into small numbers. Then, the

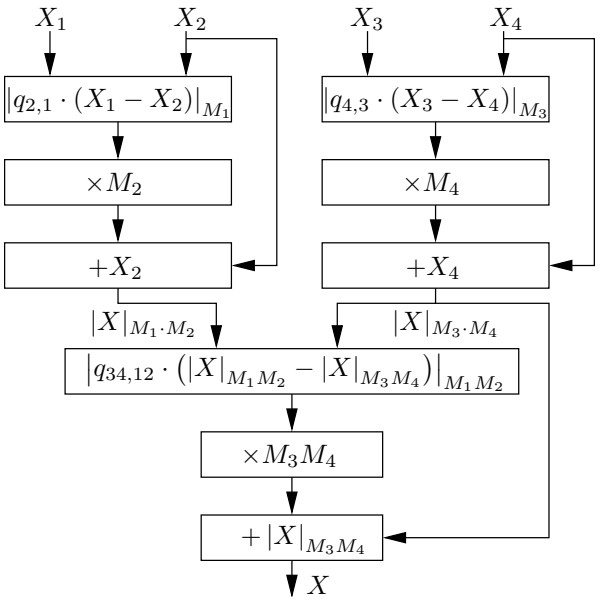


Fig. 4. Reverse converter based on the new CRT II for the 4-moduli RNS.

factors for moduli $2^k \pm 1$ are the bases for the bottom level RNSs. The ranges of the bottom level RNSs are then equal to appropriate moduli $2^k \pm 1$. The number of possible systems is limited by the number of factorisable numbers $2^k \pm 1$.

The number $2^k - 1$ is not prime, if k is not prime (Biernat, 2007), since

$$2^{ij} - 1 = (2^i - 1)(2^{0\cdot i} + 2^{1\cdot i} + 2^{2\cdot i} + \cdots + 2^{(j-1)\cdot i}). \quad (19)$$

There are also non-prime numbers $2^k - 1$ for prime k, e.g., $2^{11} - 1 = 23 \cdot 89$.

The numbers $2^k + 1$ are not prime, if k is neither prime nor a power of two (Biernat, 2007), since, for any odd j,

$$2^{ij} + 1 = (2^i + 1)(2^{0\cdot i} - 2^{1\cdot i} + 2^{2\cdot i} - \cdots + 2^{(j-1)\cdot i}). \quad (20)$$

The condition that $2^k \pm 1$ is not prime is not sufficient to build an efficient HRNS, since many non-prime numbers have large factors. As an example, consider $2^{28}+1 = 17 \cdot 15790321$. The arithmetic units (e.g., multipliers) modulo 15790321 can be larger and slower than modulo $2^{28} + 1$. The limit of the factor size for efficient implementations highly depends on arithmetic units structures and implementation technology. In Table 1 there are some factorisable numbers $2^k \pm 1$ which can be used as a base for searching the required HRNS.

Many different HRNS classes can be built using moduli from Table 1. For large dynamic ranges the moduli with small factors can be combined, i.e., $2^{50} - 1$, $2^{84} - 1$ or $2^{102} - 1$. In this paper, only HRNSs with the top level base $(2^k - 1, 2^k, 2^k + 1)$ are investigated. This approach allows using very efficient converter structures designed for the RNS $(2^k - 1, 2^k, 2^k + 1)$. Moreover, since one of the moduli is 2^k, arithmetic operations and residue generators for those moduli are very simple (Piestrak, 1994). One disadvantage of this solution is a dynamic range limit at 90 bits, because for $k > 30$ it is difficult to find a pair $2^k \pm 1$ factorizable into small numbers.

Table 1. Moduli $2^k \pm 1$ factorisable into small divisors.

Modulus	Factors	Modulus	Factors
$2^6 - 1$	7, 9	$2^6 + 1$	5, 13
$2^9 - 1$	7, 73	$2^9 + 1$	19, 27
$2^{10} - 1$	3, 11, 31	$2^{10} + 1$	25, 41
$2^{12} - 1$	5, 7, 9, 13	$2^{12} + 1$	17, 241
$2^{14} - 1$	3, 43, 127	$2^{14} + 1$	5, 29, 113
$2^{15} - 1$	7, 31, 151	$2^{15} + 1$	9, 11, 331
$2^{18} - 1$	7, 19, 27, 73	$2^{18} + 1$	5, 13, 37, 109
$2^{24} - 1$	5, 7, 9, 13, 17, 241	$2^{24} + 1$	97, 257, 673
$2^{30} - 1$	7, 9, 11, 31, 151, 331	$2^{30} + 1$	13, 25, 41, 61, 1321
	Factors		
$2^{32} - 1$	3, 5, 17, 257, 65537		
$2^{42} - 1$	9, 43, 49, 127, 337, 5419		
$2^{44} - 1$	3, 5, 23, 89, 397, 683, 2113		
$2^{50} - 1$	3, 11, 31, 251, 601, 1801, 4051		
$2^{52} - 1$	3, 5, 53, 157, 1613, 2731, 8191		
$2^{72} - 1$	5, 7, 13, 17, 19, 27, 37, 73, 109, 241, 433, 38737		
$2^{84} - 1$	5, 9, 13, 29, 43, 49, 113, 127, 337, 1429, 5419, 14449		
$2^{100} - 1$	3, 11, 31, 41, 101, 125, 251, 601, 1801, 8101, 4051, 268501		
$2^{102} - 1$	7, 9, 103, 307, 2143, 2857, 6529, 11119, 43691, 131071		

The proposed HRNS has two levels. Arithmetic computations are performed at the bottom level RNS with the base consisting of factors of $2^k \pm 1$ and 2^k. The converters can be built as hierarchical structures, where large number computations are done with converters for the RNS $(2^k - 1, 2^k, 2^k + 1)$. Additionally, the proposed HRNS allows performing some difficult operations (e.g., sign detection (Tomczak, 2008)) after partial conversion to the RNS $(2^k - 1, 2^k, 2^k + 1)$. Thus, the proposed HRNS class allows using small moduli in arithmetic channels and simple, efficient converters developed for the RNS $(2^k - 1, 2^k, 2^k + 1)$.

As an example of computations in the proposed HRNS consider the multiplication of two 44-bit binary numbers

$$X = 17592186044415_{10},$$
$$Y = 17592186044414_{10}$$

in the HRNS with the largest 90-bit dynamic range. The

88-bit result is

$$W = X \cdot Y = 309485009821292292166647810_{10}.$$

The first step is the conversion of the numbers X and Y to the RNS $(2^{30}-1, 2^{30}, 2^{30}+1)$ representations:

$$X = (16383, 1073741823, 1073725440),$$

$$Y = (16382, 1073741822, 1073725439).$$

Next, the residues modulo $2^{30}-1$ are written using the RNS $(7, 9, 11, 31, 151, 331)$, whereas the residues modulo $2^{30}+1$ are written using the RNS $(13, 25, 41, 61, 1321)$. Thus, the representations of the numbers X and Y in the bottom level RNS are $X = (3, 3, 4, 15, 75, 164,\ 1073741823,\ 8, 15, 15, 24, 788)$ and $Y = (2, 2, 3, 14, 74, 163,\ 1073741822,\ 7, 14, 14, 23, 787)$. Next, the value of $W = X \cdot Y$ is computed in the bottom level RNS according to Eqn. (4) as

$$\begin{aligned} W &= (|3 \cdot 2|_7, |3 \cdot 2|_9, |4 \cdot 3|_{11}, \ldots) \\ &= (6, 6, 1, 24, 114, 252,\ 2,\ 4, 10, 5, 3, 607). \end{aligned}$$

Then, the conversion of W to the top-level RNS $(2^{30}-1, 2^{30}, 2^{30}+1)$ results in

$$W = (268386306, 2, 268484610),$$

which is a representation of the required result $309485009821292292166647810$.

3.1. Multiplier's complexity. The reduction of the moduli width usually results in smaller, faster and less power hungry arithmetic circuits. In this section, the standard unit-gate model by Zimmermann (1999) is used for area estimation and comparison of multipliers in the 3-moduli RNS $(2^k-1, 2^k, 2^k+1)$ and in the proposed HRNS. According to this model, each two-input monotonic gate (e.g., AND, OR, NAND, NOR) has the area $A = 1$, two-input XOR and XNOR gates have the area $A = 2$ and a 1-bit full-adder has the area $A = 7$. For gates with the number of inputs $a > 2$, the area is $a - 1$ times larger than that of a single, two input gate performing the same logic function.

Multiplier area comparison requires area estimation of different types of multipliers. The areas of the k-bit binary multiplier and multipliers modulo $2^k \pm 1$ are taken from the work of Zimmermann (1999), whereas the area of a multiplier modulo any other k-bit number is estimated for the multiplier presented by Hiasat (2000). The area of a multiplier modulo 2^k is assumed as a half of a full k-bit binary multiplier area. In all multipliers, Wallace trees with no Booth reduction are used for carry-save addition and fast parallel prefix adders are used. The formulas used for area estimation are given in Table 2.

Table 2. Area estimation formulas for different multipliers using a unit-gate model. The formulas for multipliers modulo $2^k \pm 1$ are taken from the work of Zimmermann (1999), the formula for a multiplier modulo any other k-bit number is based on the results of Hiasat (2000).

Multiplier	Area
k-bit binary	$A_k = 8k^2 + 3k\lceil\log_2 k\rceil - 3k$
mod 2^k	$A_{2^k} = 0.5A_k$
mod 2^k-1	$A_{2^k-1} = 8k^2 + \frac{3}{2}k\lceil\log_2 k\rceil - 7k$
mod 2^k+1	$A_{2^k+1} = 9k^2 + \frac{3}{2}k\lceil\log_2 k\rceil + 11k$
mod k-bit	$A^*_k = \frac{5}{4}A_k + 15k + 3k\lceil\log_2 k\rceil + \frac{k^2}{4}$

Table 3. Estimated area comparison for multipliers in the RNS $(2^k-1, 2^k, 2^k+1)$ and in the proposed HRNS. In the 4-th column the difference between RNS and HRNS areas is shown. The gain in the 5-th column is computed as (RNS area − HRNS area)/RNS area · 100%.

Range [bit]	RNS area [gates]	HRNS area [gates]	Difference [gates]	Gain [%]
18	852	669	183	21
27	1886	1975	-89	-5
30	2305	1660	645	28
36	3270	2245	1025	31
42	4403	2998	1405	32
45	5033	3684	1349	27
54	7254	4619	2635	36
72	12696	6893	5803	46
90	19650	9950	9700	49

The area of a multiplier in the RNS is computed as a sum of the areas occupied by individual multipliers modulo moduli from the system base. Thus, the area of the full multiplier in the RNS $(2^k-1, 2^k, 2^k+1)$ is computed as a sum of areas occupied by multipliers modulo 2^k-1, 2^k and 2^k+1. The area of multipliers in the HRNS is computed in a similar way. For moduli which are not of the form $2^a \pm 1$, the multipliers from the work of Hiasat (2000) are used. The areas for RNSs with different dynamic ranges are compared in Table 3.

It should be noted that for all moduli the fastest multipliers with prefix adders are used. In real systems some multipliers, especially for smaller moduli, could be implemented as slower and smaller circuits, because the critical path is usually determined only by the largest modulus. Thus, additional area savings could be obtained.

As shown in Table 3, in all cases but one the multipliers in the HRNS are from 21 to 49 % smaller than the multipliers in the RNS $(2^k-1, 2^k, 2^k+1)$ with the same dynamic range. The estimated area of the HRNS multiplier is larger for the dynamic range equal to 27 bits because of a general modulo multiplier for moduli $19, 27, 73$. The multiplier presented by Hiasat (2000) cannot take advantage

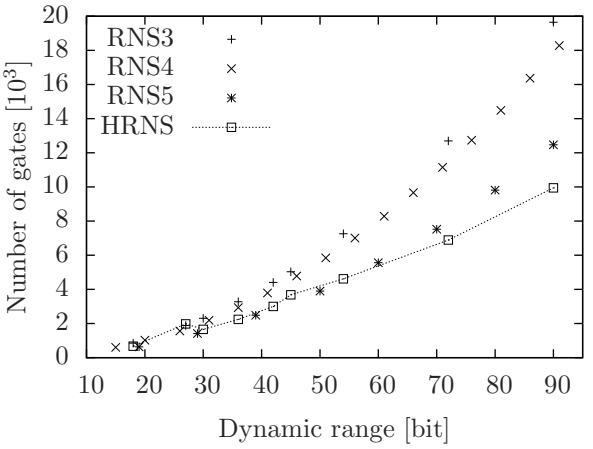


Fig. 5. Estimated multiplier area in unit gates for RNS3: the 3-moduli RNS $(2^k-1, 2^k, 2^k+1)$, RNS4: the 4-moduli RNS $(2^k-1, 2^k, 2^k+1, 2^{2k+1}-1)$, RNS5: the 5-moduli RNS $(2^k-1, 2^k, 2^k+1, 2^{k-1}-1, 2^{k+1}-1)$ and HRNS: the proposed HRNS.

of the periodicity property. The multiplier using the periodicity property could offer better parameters, because periods or half-periods for these moduli equal 9.

The complexity of the presented HRNS multipliers was also compared with that of one of the newest residue number systems offering higher parallel processing degree than the RNS $(2^k-1, 2^k, 2^k+1)$: the 4-moduli system $(2^k-1, 2^k, 2^k+1, 2^{2k+1}-1)$ presented by (Molahosseini *et al*., 2010) and the 5-moduli RNS $(2^k-1, 2^k, 2^k+1, 2^{k-1}-1, 2^{k+1}-1)$ by Cao *et al*. (2007). The results are shown in Fig 5. These moduli sets do not allow constructing systems with the same dynamic range as the proposed HRNS. Thus the comparison is done for all achievable dynamic ranges in the scope covered by the proposed HRNS. Analysing data from Fig. 5 shows that for dynamic ranges larger than 30 bits the proposed HRNS offers much smaller multipliers than for the 3- and 4-moduli RNSs, and for the dynamic ranges larger than 70 bits the multiplier area is less than for the 5-moduli RNS. Moreover, for dynamic ranges between 30 and 70 bits HRNS multipliers complexity is comparable to that of multipliers for the 5-moduli RNS, but HRNS multipliers can be used for dynamic ranges for which the 5-moduli RNS is not available. Thus, the proposed HRNS based on the 3-moduli RNS allows building multipliers with a comparable or smaller area than the 5-moduli general RNS.

3.2. Conversion from the HRNS. In this section, equations for efficient conversion from the proposed HRNS to the binary number system are described. The conversion from a new HRNS to the binary number system is performed in two steps. In the first one, the residues modulo $2^k \pm 1$ are computed from the residues for appropriate factors of $2^k \pm 1$. In the second step, a typical converter from the RNS $(2^k-1, 2^k, 2^k+1)$ is used. Since in the literature there are known many efficient converters from the RNS $(2^k-1, 2^k, 2^k+1)$ (Piestrak, 1995; Wang *et al*., 2000; Mohan, 2001; Bi *et al*., 2004), in this paper only the first step is analysed.

Because most of the reverse conversion equations proposed in this paper are based on (18), it is rewritten to allow simple and efficient implementation. Transformations are chosen to replace complex arithmetic operations (such as multiplications and residue computing) with simple logical operations on bit fields (concatenation, rotation, etc.). If complex modulo operations are difficult to implement with a simple logic, Look-Up Tables (LUTs) based on read-only memories (ROMs) are used. In this case, the reverse conversion equations are rewritten to minimize the ROM address width.

The first operation in Eqn. (18) is the difference $X_1 - X_2$. If M_1 is small, then it is desirable to compute the difference as a residue modulo M_1 or at least as a number congruent modulo M_1 to $|X_1 - X_2|_{M_1}$ and less than the difference $X_1 - X_2$. Therefore, the rest of the operations in $|q_{2,1} \cdot (X_1 - X_2)|_{M_1} \cdot M_2$ can be implemented with a simple circuit due to a low operand width. This approach gives especially good results when M_2 is much larger than M_1 and $P(M_1)$ or $HP(M_1)$ is small. Hence, the operation $|X_1 - X_2|_{M_1}$ can be replaced with

$$|X_1 - X_2|_{M_1} = \left| |X_1 - |X_2|_{2^{P(M_1)}-1}|_{2^{P(M_1)}-1} \right|_{M_1} \quad (21)$$

or

$$|X_1 - X_2|_{M_1} = \left| |X_1 - |X_2|_{2^{HP(M_1)}+1}|_{2^{HP(M_1)}+1} \right|_{M_1}. \quad (22)$$

The operations from Eqns. (21) and (22) can be implemented efficiently with circuits based on the periodicity property. First, observe that if $X_1 < 2^b$ and $X_2 < 2^b$ for some b, then

$$\begin{aligned} |X_1 - X_2|_{2^b-1} &= |X_1 + (2^b-1) - X_2 - (2^b-1)|_{2^b-1} \\ &= |X_1 + \overline{X_2}|_{2^b-1} \end{aligned} \quad (23)$$

and

$$\begin{aligned} |X_1 - X_2|_{2^b+1} &= |X_1 + (2^b-1) - X_2 - (2^b-1)|_{2^b+1} \\ &= |X_1 + \overline{X_2} + 2|_{2^b+1}, \end{aligned} \quad (24)$$

where $\overline{X_2} = (2^b-1) - X_2$ denotes bit-by-bit complementation of X_2 binary representation. Notice that only b least

significant bits of X_2 are complemented. Since $X_1 < 2^b$ for $b = P(M_1)$ or $b = HP(M_1)$ and

$$X_2 = \sum_a 2^{a \cdot b} \cdot B_a \tag{25}$$

is a concatenation of b-bit binary fields B_a defined by Eqn. (9), we have

$$\begin{aligned} |X_1 - X_2|_{M_1} &= \left| \left| X_1 + \sum_a \overline{B_a} \right|_{2^{P(M_1)}-1} \right|_{M_1} \\ &= \left| X_1 + \sum_a \overline{B_a} \right|_{M_1} \end{aligned} \tag{26}$$

or

$$\begin{aligned} &|X_1 - X_2|_{M_1} \\ &\quad = \left| \left| X_1 + \sum_{a \text{ even}} (\overline{B_a} + 2) + \sum_{a \text{ odd}} B_a \right|_{2^{HP(M_1)}+1} \right|_{M_1} \\ &\quad = \left| X_1 + \sum_{a \text{ even}} (\overline{B_a} + 2) + \sum_{a \text{ odd}} B_a \right|_{M_1} . \end{aligned} \tag{27}$$

Next, the difference $|X_1 - X_2|_{M_1}$ is multiplied modulo by the multiplicative inverse $\left|M_2^{-1}\right|_{M_1}$. If $P(M_1)$ or $HP(M_1)$ is small, then we can try to choose M_2 such that the inverse $\left|M_2^{-1}\right|_{M_1}$ is a sum of a small number of powers of 2. In this case, the multiplication modulo M_1 can be computed as a sum modulo M_1 of cyclic rotated differences $|X_1 - X_2|_{M_1}$. The ideal case is when $\left|M_2^{-1}\right|_{M_1} = 2^j$ and computations can be done modulo $2^{P(M_1)} - 1$. The modulo multiplication by a multiplicative inverse is then a simple left cyclic rotation by j bits.

The last complex operation in Eqns. (18) is the multiplication by M_2. In some of the proposed conversion equations, this multiplication is replaced with a concatenation or a sum of binary vectors representing $|q_{2,1} \cdot (X_1 - X_2)|_{M_1}$. The simplest implementation (concatenation only) is when M_2 is a sum of $\pm 2^{b_j}$ and $2^{b_{j+1}}/2^{b_j} > M_1$ for any j.

The transformed reverse conversion process illustrates the following equation for computing the residue modulo $2^{2k} - 1$ from the residues modulo $2^k \pm 1$:

$$\begin{aligned} &|X|_{2^{2k}-1} \\ &\quad = |X|_{2^k+1} \\ &\quad + \left| \underbrace{\left| \left(2^k+1\right)^{-1} \right|_{2^k-1}}_{2^{k-1}} \cdot \left(|X|_{2^k-1} - |X|_{2^k+1} \right) \cdot \right|_{2^k-1} \\ &\quad \cdot \left(2^k + 1\right). \end{aligned} \tag{28}$$

Equation (28) can be used as a conversion base for all higher level moduli $2^a - 1$, a even, e.g., $2^6 - 1$, $2^{12} - 1$, $2^{18} - 1$, etc. In (28), the difference $X_1 - X_2$ can be computed with the use of Eqn. (26), the multiplicative inverse $\left| \left(2^k + 1\right)^{-1} \right|_{2^k-1}$ is a power of two and the quotient $2^k/2^0$ (since $M_2 = 2^k + 2^0$) is larger than $M_1 = 2^k - 1$. The implementation details of operations from Eqn. (28) will be presented in Section 4.1. The reverse conversion equations for high level moduli different than $2^{2k} - 1$ are presented below.

3.2.1. Conversion for $2^6 + 1$. $2^6 + 1$ has two factors: 5 and 13. Thus the reverse conversion equation can be directly based on Eqn. (18). There are two possible versions:

$$|X|_{65} = |X|_5 + \left| \underbrace{\left|5^{-1}\right|_{13}}_{8} \cdot \left(|X|_{13} - |X|_5 \right) \right|_{13} \cdot 5 \tag{29}$$

and

$$\begin{aligned} &|X|_{65} \\ &\quad = |X|_{13} + \left| \underbrace{\left|13^{-1}\right|_5}_{2} \cdot \left(|X|_5 - |X|_{13} \right) \right|_5 \cdot 13. \end{aligned} \tag{30}$$

Since the results of $|X|_{13} - |X|_5$ and $|X|_5 - |X|_{13}$ are both 5-bit wide, a small LUT of size $2^5 \times 6$ bits can be used to compute the second operand of the main sum. In this case, both equations should result in a similar circuit complexity.

3.2.2. Conversion for $2^9 - 1$. $2^9 - 1$ has two factors: 7 and 73. Thus

$$\begin{aligned} &|X|_{511} \\ &\quad = |X|_{73} + \left| \underbrace{\left|73^{-1}\right|_7}_{5} \cdot \left(|X|_7 - |X|_{73} \right) \right|_7 \cdot 73, \end{aligned} \tag{31}$$

There is also possible the second version with calculations modulo 73, but the circuits for computations modulo 7 are usually simpler, smaller and faster than modulo 73.

3.2.3. Conversion for $2^9 + 1$. $2^9 + 1$ also has two factors 19 and 27. Thus the value mod $2^9 + 1$ is given by

$$\begin{aligned} &|X|_{513} \\ &\quad = |X|_{27} + \left| \underbrace{\left|27^{-1}\right|_{19}}_{12} \cdot \left(|X|_{19} - |X|_{27} \right) \right|_{19} \cdot 27, \end{aligned} \tag{32}$$

or

$$|X|_{513} = |X|_{19} + \left| \underbrace{\left|19^{-1}\right|_{27}}_{10} \cdot (|X|_{27} - |X|_{19}) \right|_{27} \cdot 19. \quad (33)$$

Since the bit vectors representing $|X|_{19}$ and $|X|_{27}$ have the same width and values of $q_{2,1}$ have almost the same binary representations, either equation should result in a similar implementation complexity.

3.2.4. Conversion for $2^{10}-1$. The three factors 3, 11 and 31 of $2^{10}-1$ impose the two step conversion. In the first step, the residue modulo $3 \cdot 11 = 31$ is calculated according to

$$|X|_{33} = |X|_{11} + \left| \underbrace{\left|11^{-1}\right|_{3}}_{2} \cdot (|X|_{3} - |X|_{11}) \right|_{3} \cdot 11, \quad (34)$$

and in the second step, the residue modulo $2^{10}-1$ is calculated from residues modulo 31 and 33 according to Eqn. (28). The computation of the residue modulo 33 can be also done with the second version of Eqn. (34) with moduli 11 and 3 swapped, but then the multiplication of the difference $|X|_{11} - |X|_3$ by $|3^{-1}|_{11}$ had to be done modulo 11, which has a much larger period $P(11) = 10$ and half-period $HP(11) = 5$ than $P(3) = 2$ and $HP(3) = 1$.

3.2.5. Conversion for $2^{10}+1$. Since $2^{10}+1$ has two factors 25 and 41, the conversion is done according to

$$|X|_{1025} = |X|_{41} + \left| \underbrace{\left|41^{-1}\right|_{25}}_{11} \cdot (|X|_{25} - |X|_{41}) \right|_{25} \cdot 41. \quad (35)$$

There is also the second version with computations modulo 41, but since both 25 and 41 have large periods; thus the multiplication modulo 25 of the difference by constant 11 has to be done with an LUT. The number of LUT output bits is smaller by one bit for the result modulo 25 than modulo 41.

3.2.6. Conversion for $2^{12}+1$. The two factors $17, 241$ of $2^{12}+1$ allow conversion according to

$$|X|_{4097} = |X|_{241} + \left| \underbrace{\left|241^{-1}\right|_{17}}_{6} \cdot (|X|_{17} - |X|_{241}) \right|_{17} \cdot \underbrace{241}_{11110001_2}. \quad (36)$$

Although there are complex modulo operations in (36), since they are performed modulo 17, the MOMA modulo 2^4+1 by Piestrak (1994) can be used.

3.2.7. Conversion for $2^{14}-1$. $2^{14}-1$ has three factors: 3, 43 and 127. Thus the conversion is done in two steps. In the first step the residue modulo $3 \cdot 43 = 129$ is computed according to

$$|X|_{129} = |X|_{43} + \left| \underbrace{\left|43^{-1}\right|_{2}}_{1} \cdot (|X|_{3} - |X|_{43}) \right|_{3} \cdot 43. \quad (37)$$

Next, the final value modulo $2^{14}-1$ is computed from $|X|_{129}$ and $|X|_{127}$ using Eqn. (28).

3.2.8. Conversion for $2^{14}+1$. $2^{14}+1 = 5 \cdot 29 \cdot 113$; thus the conversion is done in two steps. The chosen sequence of equations allows doing in the second step calculations on large numbers modulo a small number with a low period. First, the residue modulo $3277 = 29 \cdot 113$ is computed according to

$$|X|_{3277} = |X|_{113} + \left| \underbrace{\left|113^{-1}\right|_{29}}_{19} \cdot (|X|_{29} - |X|_{113}) \right|_{29} \cdot 113. \quad (38)$$

Both 29 and 113 have the same periods and half-periods equal to 28 and, respectively, 14. Thus for the first step of the conversion the equation with an operation modulo a smaller number is chosen. The final value modulo $2^{14}+1$ is computed according to

$$|X|_{2^{14}+1} = |X|_{3277} + \left| \underbrace{\left|3277^{-1}\right|_{5}}_{3} \cdot (|X|_{5} - |X|_{3277}) \right|_{5} \cdot 3277. \quad (39)$$

3.2.9. Conversion for $2^{15}-1$. The residue modulo $2^{15}-1$ is determined in two steps. In the first step the residue modulo $7 \cdot 151 = 1057$ is computed as

$$|X|_{1057} = |X|_{151} + \left| \underbrace{\left|151^{-1}\right|_{7}}_{2} \cdot (|X|_{7} - |X|_{151}) \right|_{7} \cdot 151. \quad (40)$$

Next, the residue modulo $2^{15}-1$ is

$$|X|_{2^{15}-1} = |X|_{1057} + \left| \underbrace{\left|1057^{-1}\right|_{31}}_{21} \cdot (|X|_{31} - |X|_{1057}) \right|_{31} \cdot \underbrace{1057}_{10000100001_2}. \tag{41}$$

All complex operations in Eqns. (40) and (41) are computed modulo $2^k \pm 1$. Since the binary representation of 1057 is 10000100001_2 and the residue modulo 31 occupies 5 bits, the multiplication of a residue modulo 31 by 1057 can be implemented as a concatenation of 3 residues modulo 31.

3.2.10. Conversion for $2^{15}+1$. The three factors of $2^{15}+1$ imply two-level conversion. The first step to find $|X|_{2^{15}+1}$ is a calculation of the residue modulo $11 \cdot 331 = 3641$ as

$$|X|_{3641} = |X|_{331} + \left| \underbrace{\left|331^{-1}\right|_{11}}_{1} \cdot (|X|_{11} - |X|_{331}) \right|_{11} \cdot 331. \tag{42}$$

Knowing $|X|_{3641}$, the residue modulo $2^{15}+1$ is given as

$$|X|_{2^{15}+1} = |X|_{3641} + \left| \underbrace{\left|3641^{-1}\right|_{9}}_{2} \cdot (|X|_{9} - |X|_{3641}) \right|_{9} \cdot 3641. \tag{43}$$

In both the equations calculations are performed modulo very small moduli. Additionally, in (43) the residues for large numbers can be computed efficiently using an MOMA modulo 2^3+1. The implementation of Eqns. (42) and (43) requires also multiplications by relatively large constants, but owing to the low width of residues modulo 9 and 11, those multiplications can be implemented with small LUTs of size $2^4 \times 12$ bits.

3.2.11. Conversion for $2^{18}+1$. The factors of $2^{18}+1$ can be grouped into pairs $5 \cdot 13 = 65$ and $37 \cdot 109 = 4033$. One of the products is equal to 2^6+1, whereas the other one is $2^{12}-2^6+2^0$. The value of $|X|_{2^{18}+1}$ is computed by the two-level circuit. The first level computes residues modulo 65 according to Eqns. (29) or (30) and modulo

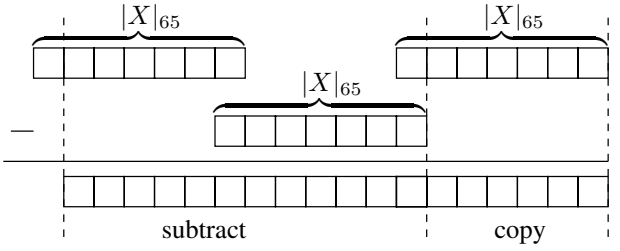


Fig. 6. Idea of the multiplication of residue modulo 65 by $4033 = 2^{12}-2^6+1$. Squares denote single bits of the residue and result. The least significant bit is on the right hand side.

4033 from

$$|X|_{4033} = |X|_{109} + \left| \underbrace{\left|109^{-1}\right|_{37}}_{18} \cdot (|X|_{37} - |X|_{109}) \right|_{37} \cdot 109. \tag{44}$$

The second version of equation for $|X|_{4033}$ (not shown) requires calculations modulo 109. The calculations modulo 109 are more complex than modulo 37 because 109 requires circuits operating on larger operands (seven bits instead of six) and the periodicity property cannot be used due to large $P(A)$ and $HP(A)$. Additionally, $\left|37^{-1}\right|_{109} = 56$ is wider than 18, which complicates multiplication by the multiplicative inverse.

Next, the value of $|X|_{2^{18}+1}$ can be computed from

$$|X|_{2^{18}+1} = |X|_{4033} + \left| \underbrace{\left|4033^{-1}\right|_{65}}_{22} \cdot (|X|_{65} - |X|_{4033}) \right|_{65} \cdot \underbrace{4033}_{111111000001_2}. \tag{45}$$

Equation (45) allows using the MOMA modulo 65, which performs operations according to (27), for computations in the right addend.

Multiplication by 4033 can be implemented as a 12-bit subtraction due to a low width of residue modulo 65 and a special form of constant $4033 = 2^{12}-2^6+2^0$. The idea is shown in Fig. 6. Notice that the result requires at most 18 bits, because $4033 \cdot 65 < 2^{18}$. The six least significant bits of the result are equal to the bits of residue modulo 65.

3.2.12. Conversion for $2^{24}+1$. $2^{24}+1$ has three factors. Thus conversion according to Eqn. (18) requires two

steps. In the first one, the residue modulo $97 \cdot 673 = 65281$ is computed as

$$|X|_{65281} = |X|_{673} + \left| \underbrace{\left|673^{-1}\right|_{97}}_{16} \cdot (|X|_{97} - |X|_{673}) \right|_{97} \cdot 673. \tag{46}$$

In a circuit based on (46), the residue generator may be necessary to compute residues modulo 97 for $|X|_{673}$ and/or for the difference. However, since $\left|673^{-1}\right|_{97}$ is a power of 2 (here, equal to 16), the multiplication of the difference can be implemented as a left shift by four bits.

After computing $|X|_{65281}$, the residue modulo $2^{24}+1$ is

$$|X|_{2^{24}+1} = |X|_{65281} + \left| \underbrace{\left|65281^{-1}\right|_{257}}_{86} \cdot (|X|_{257} - |X|_{65281}) \right|_{257} \cdot \underbrace{65281}_{1111111100000001_2}. \tag{47}$$

Multiplication by $65281 = 2^{16} - 2^8 + 2^0$ can be implemented as subtraction in a way similar to that presented in Fig. 6. The only operation modulo in Eqn. (47) is a computation of residue modulo 257 for $86 \cdot (|X|_{257} - |X|_{65281})$, which can be implemented as two MOMAs for 8-bit operands. The first MOMA computes $||X|_{257} - |X|_{65281}|_{257}$ according to Eqn. (27). The second MOMA realizes multiplication by 86 modulo 257. There is also possible the use of one MOMA, which can realise subtraction and multiplication by 86 as a sum of shifted bit fields taken from $|X|_{62581}$ and $|X|_{257}$ as shown in Fig. 7 (which does not contain additional constants equal to 2 defined by (27)). All bits set to 1 shown in Fig. 7 and additional constants can be replaced by cumulative correction equal to sum modulo 257 of all constant values. For the case shown in Fig. 7, cumulative correction (including bits set to ones) is equal to $|252+248+224+128+1+254+3+252+15+240+63+192+12 \cdot 2|_{257} = 97$.

3.2.13. Conversion for $2^{30}+1$. The reverse conversion for $2^{30}+1$ is the most complex task among those presented. In the first step, the residues modulo $25 \cdot 1321 = 33025 = 2^{15} + 2^8 + 1$ and $13 \cdot 41 \cdot 61 = 32513 = 2^{15} - 2^8 + 1$ are computed, whereas in the second one the residue modulo $2^{30}+1$ is found.

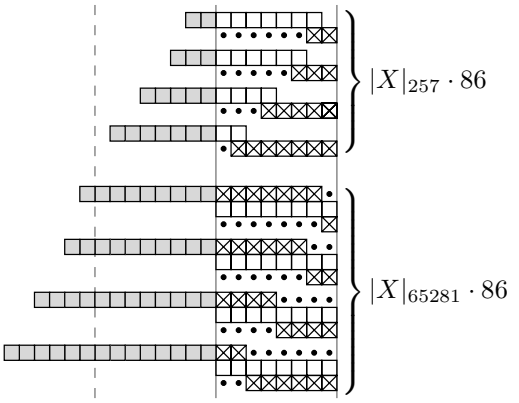


Fig. 7. Input words to the MOMA modulo 257 computing $\left|86 \cdot (|X|_{257} - |X|_{65281})\right|_{257}$. The words are constructed according to Eqn. (27). White squares denote unchanged bits, crossed squares denote complemented bits, gray squares denote bits which are wrapped, and black dots denote bits equal to 1. The least significant bits are on the right hand side. Twelve constants which are equal to 2 for each complemented bit field are not shown.

The residue modulo 33025 is computed as

$$|X|_{33025} = |X|_{1321} + \left| \underbrace{\left|1321^{-1}\right|_{25}}_{6} \cdot (|X|_{25} - |X|_{1321}) \right|_{25} \cdot 1321. \tag{48}$$

The residue modulo 32513 is determined by the residues for factors 13, 41 and 61. Thus the reverse conversion according to (18) requires a two-level circuit. There are two forms of Eqn. (18), which result in a similar complexity. In the first one, the residue modulo $41 \cdot 61 = 2501$ is computed, whereas in the second case the residue modulo $13 \cdot 61 = 793$ is found first.

The first method of reverse conversion is described by

$$|X|_{2501} = |X|_{41} + \left| \underbrace{\left|41^{-1}\right|_{61}}_{3} \cdot (|X|_{61} - |X|_{41}) \right|_{61} \cdot 41,$$

$$|X|_{32513} = |X|_{2501} + \left| \underbrace{\left|2501^{-1}\right|_{13}}_{8} \cdot (|X|_{13} - |X|_{2501}) \right|_{13} \cdot 2501. \tag{49}$$

The second method consists of the following steps:

$$|X|_{793} = |X|_{61} + \left| \underbrace{\left|61^{-1}\right|_{13}}_{3} \cdot (|X|_{13} - |X|_{61}) \right|_{13} \cdot 61, \quad (50)$$

$$|X|_{32513} = |X|_{793} + \left| \underbrace{\left|793^{-1}\right|_{41}}_{3} \cdot (|X|_{41} - |X|_{793}) \right|_{41} \cdot 793.$$

Neither form allows efficient implementation using the periodicity property. However, since operations are computed modulo 13, 41 or 61, small LUTs can be used.

After the computation of residues modulo 33025 and 32513, the residue modulo $2^{30}+1$ is computed according to the CRT,

$$|X|_{2^{30}+1} = \Big| 33025 \cdot \left|33025^{-1}\right|_{32513} \cdot |X|_{32513} + 32513 \cdot \left|32513^{-1}\right|_{33025} \cdot |X|_{33025} \Big|_{2^{30}+1}. \quad (51)$$

After transformations of Eqn. (51),

$$\begin{aligned} |X|_{2^{30}+1} &= \Big| 33025 \cdot 16193 \cdot |X|_{32513} + 32513 \cdot 16577 \cdot |X|_{33025} \Big|_{2^{30}+1} \\ &= \Big| (2^{15}+2^8+1) \cdot (2^{14}-2^8+2^6+1) \cdot |X|_{32513} + (2^{15}-2^8+1) \cdot (2^{14}+2^8-2^6+1) \cdot |X|_{33025} \Big|_{2^{30}+1}, \end{aligned} \quad (52)$$

the final reverse conversion equation is

$$|X|_{2^{30}+1} = \Big| (2^{29}-2^{21}+2^6+1) \cdot |X|_{32513} + (2^{29}+2^{21}-2^6+1) \cdot |X|_{33025} \Big|_{2^{30}+1}. \quad (53)$$

Equation (53) can be implemented using an MOMA which computes a sum of shifted bit vectors representing $|X|_{32513}$ and $|X|_{33025}$. The reverse conversion based on the CRT was used because the final addition modulo $2^{30}+1$ can be efficiently done with MOMA adding operands constructed in a way similar to that shown in Fig. 7.

Table 4. Periods and half periods for factors of $2^k \pm 1$.

M_i	HP	P	M_i	HP	P
3	1	2	251	25	50
5	2	4	257	8	16
7	–	3	307	51	102
9	3	6	331	15	30
11	5	10	337	–	21
13	6	12	397	22	44
17	4	8	433	36	72
19	9	18	601	–	25
23	–	11	673	24	48
25	10	20	683	11	22
27	9	18	1321	30	60
29	14	28	1429	42	84
31	–	5	1613	26	52
37	18	36	1801	–	25
41	10	20	2113	22	44
43	7	14	2143	–	51
49	–	21	2731	13	26
53	26	52	2857	51	102
61	30	60	4051	25	50
73	–	9	5419	21	42
89	–	11	6529	51	102
97	24	48	8101	50	100
101	50	100	8191	–	13
103	–	51	11119	–	51
109	18	36	14449	42	84
113	14	28	38737	36	72
125	50	100	43691	17	34
127	–	7	65537	16	32
151	–	15	131071	–	17
157	26	52	268501	50	100
241	12	24			

3.3. Conversion to the HRNS. Conversion from the binary number system to the HRNS can be performed in two steps. In the first one, the residues $|X|_{2^k \pm 1}$ and $|X|_{2^k}$ are computed, in the second—the residues modulo factors of $2^k \pm 1$ for $|X|_{2^k \pm 1}$ are calculated. The first step can be efficiently performed using the periodicity property. However, this step is omitted for operands smaller than 2^k, e.g., when the dynamic range of the HRNS allows performing multiplication and many additions without overflow. This situation is more likely for the HRNS with a small dynamic range, e.g., the HRNS constructed from the RNS $(2^6-1, 2^6, 2^6+1)$ allows accumulating $\approx 2^6$ results of multiplication of two 6-bit operands.

The calculation of residues modulo factors of $|X|_{2^k \pm 1}$ can be done with any residue generator. Additionally, when the periods or half periods for different factors of $2^k \pm 1$ are equal (e.g., $P(3) = HP(5) = 2$ or $P(19) = P(27) = 18$), the residue generators for this factors may share the same hardware, which allows further area reduction. The periods and half periods for factors of $2^k \pm 1$ are listed in Table 4.

Hardware sharing may be carried out in two cases. First, when for the numbers M_1 and M_2 $HP(M_1) = HP(M_2)$ or $P(M_1) = r \cdot P(M_2)$ for $r = 1, 2, 3, \ldots$. Then, when residues modulo M_1 and M_2 are calculated for the same input number X_i, two generators modulo M_1 and M_2 can have a common MOMA calculating $|X_i|_{2^{HP(M_1)}+1}$ or, respectively, $|X_i|_{2^{r \cdot P(M_2)}-1}$. Residue generators have common input when moduli M_1 and M_2 are factors of the same number $2^k \pm 1$, or when the input integer X is smaller than $2^k - 1$.

As an example consider the top level base $(2^{18} - 1, 2^{18}, 2^{18} + 1)$. The number $2^{18} - 1$ has the factor of 7, and the number $2^{18} + 1$ has the factors 5 and 13. Since $P(13) = 4 \cdot P(7) = 3 \cdot P(5)$, if $X < 2^{18} - 1$, we can use one adder modulo $2^{12} - 1$, which reduces X to a 12-bit number. Then the residues modulo 5, 7 and 13 can be computed for the 12-bit number. Even if $X > 2^{18} - 1$, the reduced 12-bit vector can be used as input for generators modulo 5 and 13, which are factors of the same modulus.

The second case when the periodicity property can simplify residue generators is for $HP(M_1) = r \cdot P(M_2)$. In this case, the residues modulo $2^{HP(M_1)} + 1$ and $2^{r \cdot P(M_2)} - 1$ are computed according to Eqns. (12) and (13). First, the input number X_i has to be partitioned into $HP(M_1)$-bit wide fields B_a. Next, three multi-operand adders should be used for the computation of $\sum_{a \text{ even}} B_a$, $\sum_{a \text{ odd}} B_a$ and $\sum_{a \text{ odd}} \overline{B_a}$. The sum $\sum_{a \text{ odd}} \overline{B_a}$ can be computed modulo $2^{HP(M_1)} + 1$. Finally, the residues $|X_i|_{2^{HP(M_1)}+1}$ and $|X_i|_{2^{r \cdot P(M_2)}-1}$ can be computed according to Eqns. (12) and (13). In such implementation the computation of $\sum_{a \text{ even}} B_a$ is common for both circuits, which results in hardware savings.

As an example consider the top level base $(2^{30} - 1, 2^{30}, 2^{30} + 1)$. The number $2^{30} - 1$ has the factors 7 and 9, the number $2^{30} + 1$ has the factor of 13. Now we have $HP(13) = P(9) = 2 \cdot P(7) = 6$. If the converted number X has the 45-bit binary representation, then three multi-operand adders can be used. The first adder computes the sum of four 6-bit wide fields $x_5 \ldots x_0$, $x_{17} \ldots x_{12}$, $x_{29} \ldots x_{24}$, $x_{41} \ldots x_{36}$, the second one computes the sum of four fields $x_{11} \ldots x_6$, $x_{23} \ldots x_{18}$, $x_{35} \ldots x_{30}$, $x_{44} \ldots x_{42}$ and the third one can be an MOMA modulo $2^6 + 1$ computing the sum of four fields $\overline{x}_{11} \ldots \overline{x}_6$, $\overline{x}_{23} \ldots \overline{x}_{18}$, $\overline{x}_{35} \ldots \overline{x}_{30}$, $\overline{x}_{44} \ldots \overline{x}_{42}$. Then, $|X|_7$ can be computed as a sum modulo 7 of outputs from the first and the second multi-operand adder, $|X|_9$ can be computed as a sum modulo 9 of outputs of the same adders, and $|X|_{13}$ can be computed as a sum modulo 13 of outputs from the first and the third multi-operand adder.

4. Implementation of reverse converters

In this section, reverse converter implementations for the proposed HRNS are presented. The circuits were described in VHDL and simulated with Cadence IUS 0611. After functional simulation the circuits were synthesized with Cadence Encounter RTL Compiler, version 07.20. The FreePDK45nm library and design flow was used (Stine *et al.*, 2005).

The converters proposed in this paper consist of two levels. The top level is responsible for conversion between a binary number and a representation in the RNS $(2^k - 1, 2^k, 2^k + 1)$, the bottom level does calculations between residues modulo $2^k \pm 1$ and the RNS with the base consisting of factors of $2^k \pm 1$. For converters from the bottom level RNS to residues modulo $2^k \pm 1$, two implementations were compared: the circuits based on equations presented in Section 3.2 and the circuits built according to the CRT. The conversion from the RNS $(2^k - 1, 2^k, 2^k + 1)$ to the binary number system was performed with two different architectures: the circuits presented by Wang *et al.* (2002) and by Bi *et al.* (2004).

The VHDL codes were written to maximize the use of optimisation algorithms built into the VHDL compiler. During an automated synthesis process, the VHDL compiler constructs the structures of inferred logic blocks (e.g., adders, multipliers, CSA trees) on the basis of parameters of supplied library cells to meet area and delay requirements. An example of the inferred circuit is a carry propagate adder (CPA). The same VHDL code results in different adder structures depending on area and time constraints. In this paper, two sets of synthesis constraints were applied. In the first case, no time constraints were imposed, therefore the circuits were synthesized to achieve the smallest area, i.e., all adders were synthesized as ripple-carry adders (RCAs). The second constraints set resulted in the fastest circuit, i.e., all adders were synthesized to achieve a minimal critical path delay. Detailed analysis of automatically generated adder structures showed that adders were constructed as hybrid structures. The part for less significant bits was a typical RCA to keep a small area, but for more significant bits fast carry propagation structures were used.

The second example of the inferred circuit is a CSA tree, which can be automatically built by RTL Compiler. The only designer task is to compose a chain of adders in the way which would allow doing many additions in parallel. Compiler generated CSAs usually have a better area and delay than structures made by hand. The main reason is that, for a hand-made CSA, it is difficult to consider parameters of individual cells from the library. Moreover, the structure of a CSA can be automatically tuned by the compiler to meet time constraints. The VHDL compiler can also merge a cascade of arithmetic operations into one CSA with all intermediary signals transformed to CSA form and with only one final CPA adder.

It should be noticed that it is difficult to write the VHDL code which can exploit compiler optimisations and at the same time creates MOMA structures identical to

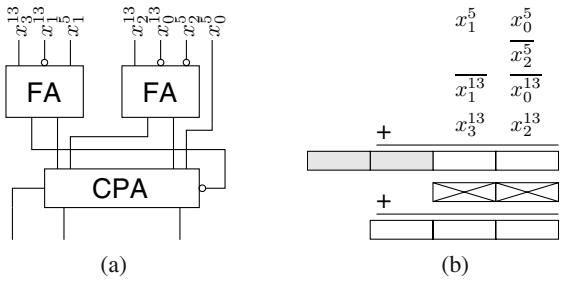


Fig. 8. Two implementations of operation $||X|_5 - |X|_{13}|_5$: strictly following Piestrak (1994) (a), VHDL description allowing automatic optimisations (b). Here x_j^M denote the j-th bit of $|X|_M$. White rectangles denote unchanged bits, crossed rectangles denote complemented bits and gray rectangles denote bits, which are wrapped. The least significant bits are on the right side. Constants equal to 2 for each complemented bit field are not shown.

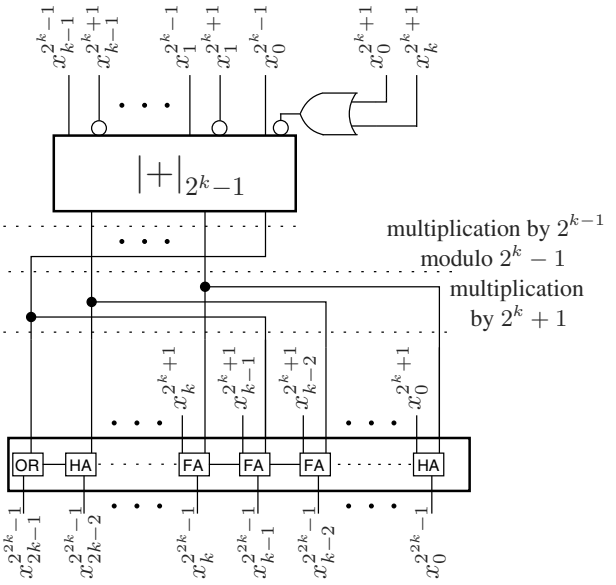


Fig. 9. Circuit to compute residue $2^{2k}-1$ from residues $2^k \pm 1$. x_j^a denote the j-th bit of binary representation of $|X|_a$.

those presented by Piestrak (1994). An example is shown in Fig. 8, where the two methods compute the result of the operation $||X|_5 - |X|_{13}|_5$ used in Eqn. (29). The circuit from Fig. 8(a) is built according to Piestrak (1994) and implemented in structural VHDL by instantiations of full-adder cells. The method from Fig. 8(b) is implemented in VHDL as two adders using the $+$ operator. The first adder computes the 4-bit sum, whereas the second one adds 2-bit fields from the computed sum. Due to different structures, the circuits differ with cumulative correction, which equals $|2+2+2+2+2|_5 = 0$ and $|2+2+2+2|_5 = 3$ for Figs. 8(a) and (b), respectively.

To check the area and the critical path delay of circuits from Fig. 8, they were used in converters built according to Eqn. (29). The area and the critical path delay for the converter with the circuit from Fig. 8(a) were 177.40 μm^2 and 899 ps for the smallest version and 286.27 μm^2 and 553 ps for the fastest version. The area and the critical path delay for the converter with the circuit from Fig. 8(b) were 169.89 μm^2 and 858 ps for the smallest version and 347.28 μm^2 and 548 ps for the fastest version. Thus, the version from Fig. 8(b) allows building smaller and faster circuits and additionally leaves more place for compiler optimisations. In this work, all MOMAs are described in the way shown in Fig. 8(b).

In residue arithmetic circuits there often occur complex computations, e.g., residue computations for moduli, which have large periods. One of the most widely known methods for the implementation of complex arithmetic operations on small arguments is the use of ROM-based LUTs. Proper implementation of ROM requires designing layout masks by hand or using automated generators supplied with a standard cell library. Unfortunately, for many free standard cell libraries there are no such generators. Moreover, after designing the layout mask, a simulation is necessary to find the delay and power of the created memory. Simulated parameters are then used to approximate full circuit characteristics. This process is complex and time consuming, especially when many ROMs are needed.

The reverse converters compared in this work require many LUTs. To avoid a long time development process, in this comparison ROMs are described in a high level language (VHDL) as combinatorial circuits consisting of a row decoder, a column decoder and a connection matrix. VHDL codes for ROMs are generated automatically based on ROM data. The area and time of generated circuits are much better than for strictly combinatorial implementations of the ROM. The test synthesis of the ROM consisting of $2^{10} \times 6$ bits gives the following results: the pure combinatorial implementation has the area 4010 μm^2 and the delay 1.8 ns (20 logic levels), the implementation with a row and column decoder results in the area 1079 μm^2 and delay 0.769 ns (6 logic levels). The above generated ROMs are used in both converters based on the proposed equations and converters according to the CRT.

4.1. Conversion circuits. For the proposed HRNS class, it is difficult to give a general reverse converter structure, because for each k the factors of $2^k \pm 1$ can be grouped in many ways. Therefore, in this section, synthesized reverse converter circuits are individually described.

Despite the circuit differences, in many of the proposed conversion circuits there is one common operation: the calculation of the residue modulo $2^{2k}-1$ from residues modulo $2^k \pm 1$. The circuit used in this paper implements

Eqn. (28), as shown in Fig. 9. Its main advantage is the lack of multipliers, which are replaced with a left cyclic rotation and a concatenation.

The first operation of the circuit from Fig. 9 is the calculation of $\left||X|_{2^k+1}\right|_{2^k-1}$. Since $|X|_{2^k+1} \leq 2^k$, the residue modulo 2^k-1 can be computed by OR-ing the least and the most significant bits of $|X|_{2^k+1}$, because for a number less than 2^k+1 only one of these bits can be equal to 1. Next, the subtraction is replaced with the addition of the additive inverse of $\left||X|_{2^k+1}\right|_{2^k-1}$ modulo 2^k-1, which is nothing else but

$$\left|2^k-1-\left||X|_{2^k+1}\right|_{2^k-1}-\left(2^k-1\right)\right|_{2^k-1}$$

and can be computed as the bit by bit complementation of $\left||X|_{2^k+1}\right|_{2^k-1}$. If the output of the adder modulo 2^k-1 at the top of Fig. 9 is without double zero representation, the rest of the circuit can be very simple, because the remaining multiplications from Eqn. (28) can be implemented with simple bit operations as shown in the middle of Fig. 9. Thus, depending on the required area and speed, the adder can be implemented as RCA with EAC and an additional double-zero elimination circuit (Biernat, 2007) or as a parallel prefix adder, e.g., the one presented by Zimmermann (1999). The final operation is the addition of the number $|X|_{2^k+1}$ to

$$\left|2^{k-1}\cdot\left(|X|_{2^k-1}-|X|_{2^k+1}\right)\cdot\right|_{2^k-1}\cdot\left(2^k+1\right)$$

implemented with the adder at the bottom of Fig. 9. The final adder can be realised with any architecture (e.g., as a parallel prefix adder), but in Fig. 9 the RCA is used to show that the most significant $k-1$ cells of the adder are used only for carry propagation. Thus, the total area of the final adder can be lowered comparing to a full $2k$-bit adder.

The circuits based on the architecture shown in Fig. 9 are used for the reverse conversion for moduli 2^6-1, $2^{12}-1$, $2^{18}-1$, $2^{24}-1$ and for $2^{30}-1$. Converters for other moduli will be described below.

The converter for 2^6+1 is based on Eqn. (30). First, the two 2-bit fields from residue $|X|_5$ and two 2-bit fields of residue $|X|_{13}$ are added using an MOMA modulo 2^6+1 of Fig. 8(b). The constant, cumulative correction (here equal to 3) required in the MOMA is not added in this step to reduce the result width. The 3-bit output from the MOMA is next connected with an input of a combinatorial circuit synthesized using the truth table. The realized function is the addition of correction required in the MOMA and all necessary computations to obtain $|2\cdot(|X|_5-|X|_{13})|_5\cdot 13$. The last step is the addition of $|X|_{13}$ to get the final result of $|X|_{2^6+1}$.

The converter for 2^9-1 is built according to Eqn. (31). First, the difference $|X|_7-|X|_{73}$ is computed using an MOMA as a 4-bit number congruent modulo 7 to $|X|_7-|X|_{73}$. The rest of the operations from $|5\cdot(|X|_7-|X|_{73})|_7\cdot 73$ are computed using the 4-input combinatorial circuit, whose output is then added to $|X|_{73}$ to calculate $|X|_{511}=|X|_{73}+|5\cdot(|X|_7-|X|_{73})|_7\cdot 73$.

The converter for 2^9+1 is built according to Eqn. (32). Since both $P(19)=18$ and $HP(19)=9$ are large compared with an operand width (5 bits), the converter consists of a subtractor computing $|X|_{19}-|X|_{27}$, $2^6\times 5$ bit LUT for the rest of operations in $|12\cdot(|X|_{19}-|X|_{27})|_{19}$, the final multiplier by 27 and an 8-bit adder.

The converter for $2^{10}-1$ is built according to Eqns. (34) and (28). First, the difference $||X|_3-|X|_{11}|_3$ is computed using MOMA modulo 2^2-1 adding 2-bit wide fields of $|X|_3$ and $|X|_{11}$. The 3-bit wide MOMA output is then connected to inputs of a small (3-input, 5-output) combinatorial circuit performing the rest of operations from (34) except the addition of $|X|_{11}$. The combinatorial circuit is synthesized using a truth table. The residue modulo 33 is obtained after adding $|X|_{11}$ to the output of the combinatorial circuit. Next, the circuit from Fig. 9 is used to compute the final residue modulo $2^{10}-1$ from $|X|_{31}$ and $|X|_{33}$.

The converter modulo $2^{10}+1$ is built according to Eqn. (35). First, the difference $|X|_{25}-|X|_{41}$ is computed as a 7-bit U2 number. Next, the rest of operations from $|11\cdot(|X|_{25}-|X|_{41})|_{25}$ are computed using a $2^7\times 5$ bit ROM. The final residue modulo $2^{10}+1$ is obtained after the multiplication of ROM output by constant 41 and the addition of $|X|_{41}$.

The converter for $2^{12}+1$ is built according to Eqn. (36). First, the value of $|6\cdot(|X|_{17}-|X|_{241})|_{17}$ is computed using an MOMA modulo 17. Next, a multiplier by constant 241 is used, and the final addition of multiplier output and $|X|_{241}$ is done using a 13-bit adder.

The converter for $2^{14}-1$ is built according to Eqns. (37) and (28). First, the difference $||X|_3-|X|_{43}|_3$ is computed using an MOMA modulo 2^2-1. Then, it is multiplied by constant 43, and $|X|_3$ is added to get $|X|_{129}$. The residue modulo $2^{14}-1$ is finally computed from $|X|_{127}$ and $|X|_{129}$ using the circuit of Fig. 9.

The converter for $2^{14}+1$ is built according to Eqns. (38) and (39). Equation (38) is implemented as a three-level circuit. First, the difference $|X|_{29}-|X|_{113}$ are computed as an 8-bit U2 number. Next, the rest of calculations in $|19\cdot(|X|_{29}-|X|_{113})|_{29}$ is performed using a $2^8\times 5$ bit ROM. The ROM output is then multiplied by constant 113 and $|X|_{113}$ is added to the multiplier output to get $|X|_{3277}$. After obtaining $|X|_{3277}$, the circuit based on (39) is used. First, it computes $||X|_5-|X|_{3277}|_5$ using an MOMA consisting of two parts. The first part performs computations modulo 2^4-1, the second part reduces 5-bit output from the first part to 3-bit number modulo 2^2+1 by adding 2-bit wide fields of the 5-bit output. No necessary corrections resulting from (24) are built into the CSA tree forming an MOMA, thus the resulting a 3-bit vector

is next encoded using 3-input 3-output combinatorial circuit, which is synthesized using the truth table. Finally, the combinatorial circuit output is multiplied by constant 3277, and $|X|_{3277}$ is added to the multiplier output to get the residue modulo $2^{14}+1$.

The converter for $2^{15}-1$ is built according to Eqns. (40) and (41). First, the difference $|X|_7 - |X|_{151}$ is computed using an MOMA as a 4-bit number congruent modulo 7 to $|X|_7 - |X|_{151}$. Next, multiplication by 2 modulo 7 and the multiplication by 151 are done using a 4-input combinatorial circuit synthesized according to a suitable truth table. The residue $|X|_{1057}$ is obtained after the addition of $|X|_{151}$ to the output of the combinatorial circuit using a 10-bit adder. In the next stage, the difference $|X|_{31} - |X|_{1057}$ is computed using an MOMA modulo 31 as a 7-bit number congruent modulo 31 to $|X|_{31} - |X|_{1057}$. The following multiplication by 21 modulo 31 is calculated using the 7-bit input 5-bit output ROM. Since $1057 = 10000100001_2$, the multiplication of $|21 \cdot (|X|_{31} - |X|_{1057})|_{31}$ by 1057 is implemented as a concatenation of three ROM outputs. The final addition of the concatenation result and $|X|_{1057}$ is done with a 15-bit adder, which generates the unbiased result.

The converter for $2^{15}+1$ is based on Eqns. (42) and (43). The difference $||X|_{11} - |X|_{331}|_{11}$ is computed using an MOMA as a 6-bit number congruent modulo $2^{HP(11)}+1 = 2^5+1$ to $|X|_{11} - |X|_{331}$. The 6-bit wide MOMA output feeds then the ROM, which calculates the rest of operations from $|2 \cdot (|X|_{11} - |X|_{331})|_{11} \cdot 331$. The residue $|X|_{3641}$ is obtained after the addition of 12-bit ROM output (the result of the multiplication of a residue modulo 11 by 331) and $|X|_{331}$. Next, an MOMA modulo 9 is used to calculate a 4-bit number congruent to $||X|_9 - |X|_{3641}|_9$. Then, the 4-bit MOMA output feeds a combinatorial circuit which calculates the value of $|2 \cdot (|X|_9 - |X|_{3641})|_9 \cdot 3641$. The final result is obtained after the addition of $|X|_{3641}$ using a 15-bit adder.

The converter for $2^{18}+1$ is built according to Eqns. (30), (44) and (45). For the residue modulo 2^6+1, the circuit described before is used. The residue modulo $37 \cdot 109 = 4033$ is computed by the circuit based on (44). The first level computes the difference $|X|_{37} - |X|_{109}$ as an 8-bit U2 vector using a 7-bit binary subtractor. Next, the difference is partitioned into two nibbles, and two $2^4 \cdot 6$ bit ROMs are used to multiply low and high nibbles by 18 modulo 37. The multiplication results are then added in an adder modulo 37, the result of addition is multiplied by 109 and, finally, $|X|_{109}$ is added to get $|X|_{4033}$. The obtained residue modulo 4033 is then subtracted from $|X|_{65}$ using a CSA that implements Eqn. (27). Next, a second CSA is used to multiply the result by 22 modulo 65. The multiplication result is then multiplied by 4033. The final residue modulo $2^{18}+1$ is obtained after the addition of $|X|_{4033}$ to the result of the multiplication by 4033.

The converter for $2^{24}+1$ is based on Eqns. (46) and (47). First, the difference $|X|_{97} - |X|_{673}$ is computed as an 11-bit U2 vector. Next, the values of two 4-bit fields containing eight most significant bits of the vector are multiplied by 16 modulo 97 with two 4-input ROMs, one ROM for each field. For the rest of the vector (three least significant bits), the multiplication by 16 modulo 97 is done using a 3-input combinatorial circuit. The three results of multiplication are then added using an MOMA modulo 97. Next, the result of the modulo addition is multiplied by 673, and $|X|_{673}$ is added to get $|X|_{65281}$. Having obtained $|X|_{65281}$, the residue modulo $|X|_{2^{24}+1}$ is computed according to Eqn. (47). First, the difference $|X|_{257} - |X|_{65281}$ is computed from (27) as a 10-bit vector congruent modulo 257 to $||X|_{257} - |X|_{65281}|_{257}$. The result is then multiplied by 86 modulo 257 by adding rotated and/or complemented bit fields in a CSA tree built on the basis of the periodicity property. The CSA output represents the value of $|86 \cdot (|X|_{257} - |X|_{65281})|_{257}$. The residue modulo $2^{24}+1$ is obtained after the multiplication of the CSA output by 65281 and the addition of $|X|_{65281}$ to the result of multiplication by 65281.

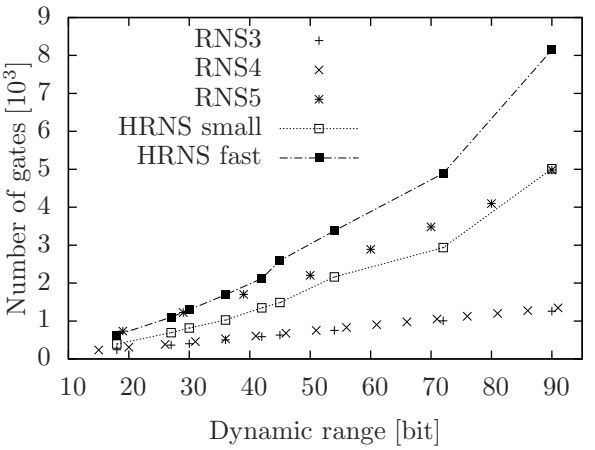


Fig. 10. Size in unit gates of reverse converters for RNS3: the smallest implementation for the 3-moduli RNS $(2^k-1, 2^k, 2^k+1)$, RNS4: the 4-moduli RNS $(2^k-1, 2^k, 2^k+1, 2^{2k+1}-1)$, RNS5: the 5-moduli RNS $(2^k-1, 2^k, 2^k+1, 2^{k-1}-1, 2^{k+1}-1)$ and for fast HRNS—the fastest and small HRNS—the smallest implemented converters for the proposed HRNS.

The converter for $2^{30}+1$ is based on Eqns. (53), (49) and (48). First, the circuits that realize Eqns. (49) and (48) in parallel compute residues modulo 32513 and 33025, which are then multiplied by constants $(2^{29}-2^{21}+2^6+1)$ and $(2^{29}+2^{21}-2^6+1)$ and added with an MOMA to get the residue modulo $2^{30}+1$ (as shown in Eqn. (53)).

The circuit computing residue modulo 32513 is built according to Eqn. (49). In the first step, the difference $|X|_{61} - |X|_{41}$ is computed as a U2 vector. The value encoded in three most significant bits of the vector is then

multiplied by 3 modulo 61 using a 3-input combinatorial circuit. The output from the circuit is added modulo 61 to the four remaining bits of the difference multiplied by 3 giving the value of $|3 \cdot (|X|_{61} - |X|_{41})|_{61}$. $|X|_{2501}$ is obtained after the multiplication of this value by 41 and the addition of $|X|_{41}$. The next step is to find the residue modulo 32513. First, the value of $8 \cdot (|X|_{13} - |X|_{2501})$ is computed using a CSA according to Eqn. (27). The result is an 8-bit number congruent modulo $2^{HP(13)+1} = 2^6 + 1$ to $|8 \cdot (|X|_{13} - |X|_{2501})|_{13}$. Next, the four most significant bits of the result are reduced modulo 13 using a 4-input ROM, and the obtained value is added to the remaining four bits with an adder modulo 13. The adder output is multiplied by 2051, and $|X|_{2501}$ is added to the multiplication result to get the final residue modulo 32513.

The residue modulo 33025 is computed according to Eqn. (48) in the parallel channel to the circuit computing $|X|_{32513}$. First, the difference $|X|_{25} - |X|_{1321}$ is computed as a 12-bit U2 number which is partitioned into three 4-bit wide fields, which are multiplied by 6 modulo 13 using three 4-input ROMs. The ROM outputs are added with an adder modulo 25, whose output is multiplied by 1321. Finally, $|X|_{1321}$ is added to the multiplication result to get the residue modulo 33025.

Table 5. Size in standard unit-gates of converters from the RNS $(2^k - 1, 2^k, 2^k + 1)$ and of the proposed HRNS converters. The overhead in the last two columns is defined as the difference between the number of gates required for HRNS converters and for the RNS converter.

Range	RNS		HRNS		Overhead	
[bit]	fast	small	fast	small	fast	small
18	504	243	628	393	124	150
27	712	366	1102	693	390	327
30	836	401	1293	815	457	414
36	947	502	1694	1025	747	523
42	1095	586	2119	1348	1024	762
45	1220	628	2591	1487	1371	859
54	1507	754	3380	2163	1873	1409
72	2050	1006	4889	2934	2839	1928
90	2796	1258	8146	5013	5350	3755

4.2. Implementation results. The area and critical path delay of the synthesized circuits are reported in Tables 6 and 7. First, the two implementations of the first step of reverse conversion are compared: the circuits proposed in this paper and converters based on the CRT. Next, full reverse converters are built by adding circuits that realize the conversion from the 3-moduli RNS $\left(2^k - 1, 2^k, 2^k + 1\right)$. The converters for the RNS $\left(2^k - 1, 2^k, 2^k + 1\right)$ were chosen from two implementations: converters presented by Bi *et al.* (2004) and by Wang *et al.* (2002). Since the area differences between these two circuits are small (Table 7), the faster circuit was used. For full, two step conversion, the comparison with CRT implementation was not done, because the converters for the RNS $\left(2^k - 1, 2^k, 2^k + 1\right)$ are superior in terms of the area and delay compared to general converters based on CRT due to the lack of ROMs and the multi-operand adder modulo the RNS dynamic range.

The parameters from Table 6 show that for the proposed HRNS class very efficient converters can be built on the basis of the proposed equations. In all cases the converters proposed in this paper are much smaller and faster than implementations based on the CRT. Thus, the proposed HRNS allows significant lowering the moduli width with low conversion overhead compared to generic converters based on the CRT.

Compared with converters for the RNS $\left(2^k - 1, 2^k, 2^k + 1\right)$ by Wang *et al.* (2002), our converters require additional hardware resources. However, the total efficiency of the residue circuit depends on arithmetic operations performance. Thus, the converter overhead is compared with area savings for multipliers, reported earlier in Table 3. The complexity of converters is computed with a unit-gate model as in Section 3.1. Due to irregular structures of HRNS converters, the area of all converters is computed on the basis of netlists generated by the Cadence RTL compiler. Each converter was synthesized in two versions: the smallest and the fastest, thus for each converter two netlists exist. For each netlist the number of the employed library cells of every type was extracted. Next, for each cell type the number of basic gates was assigned based on the realised function, e.g., for the AOI21X1 cell ($not\ ((A\ and\ B)\ or\ C)$) the assumed area was 2. The buffers and inverters were not taken into account. The computed converters area in unit gates is shown in Table 5.

The comparison of the area overhead from Table 5 and multiplier area savings from Table 3 shows that only in two cases (for 45- and 27-bit dynamic range) that the additional converter area is larger than the area savings in one multiplier. Thus, despite additional conversion cost, the proposed HRNS allows reducing the total area due to significant decrease in the multiplier area comparing to the RNS $\left(2^k - 1, 2^k, 2^k + 1\right)$.

The HRNS converter area was also compared with the areas of converters for the 4-moduli RNS $(2^k - 1, 2^k, 2^k + 1, 2^{2k+1} - 1)$ and the 5-moduli RNS $(2^k - 1, 2^k, 2^k + 1, 2k - 1 - 1, 2^{k+1} - 1)$ presented by Molahosseini *et al.* (2010) and Cao *et al.* (2007). According to Molahosseini *et al.* (2010), the area of the 4-moduli converter was assumed to be $74k + 14$ and the area for the 5-moduli converter was $5\frac{5}{6}k^2 + 162\frac{1}{6}k + \frac{7}{6}a - 7$, where $a = k - 4, 9k - 12$ and $5k - 8$ for $k = 6i - 2, 6i$ and $6i + 2$, respectively. The results are shown in Fig. 10.

HRNS converters are larger than those for the 3- and the 4-moduli RNS. However, the smallest version of HRNS converters requires less gates than converters for

Table 6. Area [μm^2] and critical path delay [ns] for output converters from the RNS defined by factors of $2^k \pm 1$.

RNS range	CRT converters				Proposed converters			
	smallest		fastest		smallest		fastest	
	area	delay	area	delay	area	delay	area	delay
2^6-1	555.65	1.522	690.34	1.078	100.90	0.978	213.06	0.423
2^6+1	527.02	1.448	643.88	0.940	169.89	0.858	347.28	0.548
2^9-1	1543.06	3.353	2125.93	2.004	208.84	1.333	460.38	0.654
2^9+1	1312.63	2.617	1717.17	1.586	485.26	1.702	665.47	0.935
$2^{10}-1$	1134.30	2.680	1724.68	1.442	322.41	2.162	694.56	1.013
$2^{10}+1$	1505.98	3.130	2294.88	1.854	603.52	1.873	890.26	1.143
$2^{12}-1$	1608.29	3.439	2605.08	1.703	482.91	2.522	1018.85	1.256
$2^{12}+1$	1816.19	5.138	3088.93	2.265	405.84	2.160	970.04	1.060
$2^{14}-1$	1841.06	4.339	3295.42	1.925	447.71	3.248	1085.96	1.348
$2^{14}+1$	2401.88	5.352	4065.55	2.386	1033.40	4.481	2033.95	2.235
$2^{15}-1$	2459.13	5.066	4371.53	2.116	788.89	4.094	1786.16	2.067
$2^{15}+1$	2278.92	6.141	4025.66	2.544	831.60	4.605	1824.64	1.986
$2^{18}-1$	3509.89	6.022	5863.43	2.515	1006.18	3.951	2055.53	1.658
$2^{18}+1$	3227.85	6.704	5872.35	2.586	1512.08	5.739	3381.78	2.665
$2^{24}-1$	4547.05	7.250	7732.19	2.527	1352.52	5.013	3251.78	1.978
$2^{24}+1$	4440.99	8.817	8804.07	3.168	2135.78	7.216	4276.73	3.034
$2^{30}-1$	5840.91	8.035	10954.40	2.504	2257.80	8.464	5848.89	2.862
$2^{30}+1$	6412.98	10.043	11075.48	3.257	4524.52	8.103	8307.55	3.527

Table 7. Area [μm^2] and critical path delay [ns] for output converters from the RNS $(2^k-1, 2^k, 2^k+1)$ and for full converters from the proposed HRNS.

k	Converter (Wang *et al*., 2002)				Converter (Bi *et al*., 2004)				Proposed converter			
	smallest		fastest		smallest		fastest		smallest		fastest	
	area	delay	area	delay	area	delay	area	delay	area	delay	area	delay
6	426.1	2.35	1162.0	0.67	508.7	2.71	1334.2	1.07	695.0	3.01	1571.7	1.28
9	637.3	3.22	1780.5	0.82	800.6	3.56	2049.0	1.13	1337.5	4.38	2926.6	1.74
10	686.1	3.30	2084.6	0.85	867.7	3.92	2447.4	1.17	1605.9	5.32	3379.4	2.03
12	788.9	4.48	2355.0	0.92	1032.9	4.41	2787.2	1.30	1724.7	6.32	4166.5	2.20
14	915.1	5.22	2725.7	0.99	1204.2	5.33	3444.7	1.31	2353.5	9.15	5525.1	3.22
15	980.4	5.57	3057.0	1.01	1300.0	5.77	3703.3	1.35	2621.0	9.47	6800.2	2.97
18	1175.6	6.80	3833.7	1.05	1551.0	6.83	4802.8	1.45	3822.5	11.52	8591.9	3.73
24	1570.3	8.96	5182.5	1.15	2090.3	9.32	6092.9	1.59	4987.2	14.99	12209.8	4.11
30	1949.9	11.10	6878.5	1.21	2610.7	11.37	8293.9	1.66	8976.3	19.15	20232.5	4.76

the 5-moduli RNS, although the number of moduli in the HRNS is equal to 5 for dynamic ranges less than 2^{30} and larger than 5 for all RNSs with the dynamic range $\geq 2^{30}$. For example, the converter for the 90-bit HRNS with 12 moduli takes almost the same area as the converter for the 5-moduli RNS with the same dynamic range. This result shows that converters for the proposed HRNS have much better area characteristics than highly optimized converters for RNSs with moduli of the form 2^k and $2^k \pm 1$.

The fastest HRNS converters are larger than those for the 5-moduli RNS, but for many HRNSs there are no 5-moduli RNSs with similar dynamic ranges. Moreover, for some 5-moduli RNSs with a similar dynamic range the area overhead of the fastest HRNS converter can be offset by replacing 5-moduli multipliers with HRNS multipliers. For example, for the 90-bit dynamic range the fastest HRNS converter is larger by 3169 logic gates and the HRNS multiplier is smaller by 2522 logic gates. Thus, the converter area overhead can be off-set by only two multipliers.

It is worth noting that for the 90-bit dynamic range the area of the smallest converter for the proposed HRNS is similar to that of the converter for the 5-moduli RNS. Thus, with comparable conversion cost, the HRNS offers much more parallelism (12 moduli instead 5). This also proves that the proposed idea of the multi-level HRNS with a very simple top-level converter based on moduli $2^k, 2^k \pm 1$ results in a large dynamic range and a high degree of parallel processing, allowing high performance of the arithmetic circuit while maintaining low converter cost.

5. Conclusion

In this work, the analysis of the hierarchical residue number system with the top level RNS $(2^k-1, 2^k, 2^k+1)$ was presented, and detailed reverse converters structures were given. The HRNS is constructed by representing residues modulo $2^k \pm 1$ in RNSs with the base consisting of factors $2^k \pm 1$. The analysis of estimated multipliers area showed that the proposed HRNS results in reducing the multipliers area up to 49% compared to multipliers for the RNS $(2^k-1, 2^k, 2^k+1)$ and up to 48% over the 4-moduli RNS presented by Molahosseini *et al*. (2010). The HRNS multipliers area is comparable to or smaller up to 20% than the area of the multipliers for the 5-moduli RNS presented by Cao *et al*. (2007). Additionally, the HRNS can be used for dynamic ranges for which the 5-moduli RNS cannot be constructed.

The proposed reverse converters are larger than those for the 3-moduli general RNS $(2^k-1, 2^k, 2^k+1)$, although in all cases but two the converter area overhead is off-set by single multiplier area savings. The proposed converters can be built as circuits smaller than converters for the 5-moduli general RNS by Cao *et al*. (2007) while in the HRNS more moduli can be used (up to 12), allowing additional savings on arithmetic circuits. Another advantage of the proposed HRNS is the possibility to perform complex arithmetic operations (e.g., sign detection) after partial conversion to the RNS $\left(2^k-1, 2^k, 2^k+1\right)$. The only limitation of the presented HRNS class is the number of dynamic ranges, but the proposed idea of replacing moduli $2^k \pm 1$ with their factors can also be applied to other RNS sets, although some more research is required.

Acknowledgment

The author is greatly indebted to Prof. Janusz Biernat for his strong support during the creation of this work, to Dr. Piotr Patronik for stimulating conversations and a lot of helpful comments, and to the anonymous reviewers for many detailed remarks concerning the overall quality of the paper.

References

Akušskij, I.J. and Judickij, D.I. (1968). *Machine Arithmetic in Residual Classes*, Sovetskoje Radio, Moscow, (in Russian).

Bi, S., Wang, W. and Al-Khalili, A. (2004). Modulo deflation in $(2^n+1, 2^n, 2^n-1)$ converters, *Proceedings of the 2004 International Symposium on Circuits and Systems, ISCAS '04, Vancouver, BC, Canada*, Vol. 2, pp. 429–432.

Biernat, J. (2007). *Architecture of Residue Arithmetic Circuits*, Exit, Warsaw, (in Polish).

Cao, B., Chang, C.-H. and Srikanthan, T. (2003). An efficient reverse converter for the 4-moduli set $\left(2^n-1, 2^n, 2^n+1, 2^{2n}+1\right)$ based on the new Chinese remainder theorem, *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* **50**(10): 1296–1303.

Cao, B., Chang, C.-H. and Srikanthan, T. (2007). A residue-to-binary converter for a new five-moduli set, *IEEE Transactions on Circuits and Systems I: Regular Papers,* **54**(5): 1041–1049.

Chokshi, R., Berezowski, K.S., Shrivastava, A. and Piestrak, S.J. (2009). Exploiting residue number system for power-efficient digital signal processing in embedded processors, *Proceedings of the 2009 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems, CASES '09, Grenoble, France*, pp. 19–28.

Conway, R. and Nelson, J. (2004). Improved RNS FIR filter architectures, *IEEE Transactions on Circuits and Systems II: Express Briefs* **51**(1): 26–28.

Hiasat, A.A. (2000). New efficient structure for a modular multiplier for RNS, *IEEE Transactions on Computers* **49**(2): 170–174.

Mohan, P.V.A. (2001). Comments on "Breaking the $2n$-bit carry propagation barrier in residue to binary conversion for the $[2^n-1, 2^n, 2^n+1]$ moduli set", *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* **48**(8): 1031.

Mohan, P.V. (2002). *Residue Number Systems: Algorithms and Architectures*, Kluwer Academic Publishers, Norwell, MA.

Molahosseini, A., Navi, K., Dadkhah, C., Kavehei, O. and Timarchi, S. (2010). Efficient reverse converter designs for the new 4-moduli sets $2^n-1, 2^n, 2^n+1, 2^{2n+1}-1$ and $2^n-1, 2^n+1, 2^{2n}, 2^{2n}+1$ based on new CRTs, *IEEE Transactions on Circuits and Systems I: Regular Papers* **57**(4): 823–835.

Piestrak, S.J. (1994). Design of residue generators and multioperand modular adders using carry-save adders, *IEEE Transactions on Computers* **43**(1): 68–77.

Piestrak, S.J. (1995). A high-speed realization of a residue to binary number system converter, *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* **42**(10): 661–663.

Piestrak, S. and Berezowski, K. (2008a). Design of residue multipliers-accumulators using periodicity, *Proceedings of the IET Irish Signals and Systems Conference, ISSC 2008, Galway, Republic of Ireland*, pp. 380–385.

Piestrak, S.J. and Berezowski, K.S. (2008). Architecture of efficient RNS-based digital signal processor with very low-level pipelining, *Proceedings of the IET Irish Signals and Systems Conference, ISSC 2008, Galway, Republic of Ireland*, pp. 127–132.

Skavantzos, A. and Abdallah, M. (1999). Implementation issues of the two-level residue number system with pairs of conjugate moduli, *IEEE Transactions on Signal Processing* **47**(3): 826–838.

Soderstrand, M.A., Jenkins, W.K., Jullien, G.A. and Taylor, F.J. (Eds.) (1986). *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*, IEEE Press, Piscataway, NJ.

Stine, J. E., Grad, J., Castellanos, I., Blank, J., Dave, V., Prakash, M., Iliev, N. and Jachimiec, N. (2005). A framework for high-level synthesis of system-on-chip designs, *Proceedings of the International Conference on Microelectronic Systems Education, Anaheim, CA, USA*, pp. 11–12.

Tomczak, T. (2008). Fast sign detection for RNS $(2^n - 1, 2^n, 2^n + 1)$, *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* **55**(6): 1502–1511.

Wang, W., Swamy, M.N.S., Ahmad, M.O. and Wang, Y. (2000). A high-speed residue-to-binary converter for three-moduli $\left(2^k, 2^k - 1, 2^{k-1} - 1\right)$ RNS and a scheme for its VLSI implementation, *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* **47**(12): 1576–1581.

Wang, W., Swamy, M.N.S., Ahmad, M.O. and Wang, Y. (2003). A study of the residue-to-binary converters for the three-moduli sets, *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* **50**(2): 235–243.

Wang, W., Swamy, M.N.S. and Ahmad, M.O. (2004). RNS application for digital image processing, *Proceedings of the 4th IEEE International Workshop on System-on-Chip for Real-Time Applications, IWSOC'04, Banff, Alberta, Canada*, pp. 77–80.

Wang, Y. (2000). Residue-to-binary converters based on new chinese remainder theorems, *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing* **47**(3): 197–205.

Wang, Y., Song, X., Aboulhamid, M. and Shen, H. (2002). Adder based residue to binary number converters for $(2^n - 1, 2^n, 2^n + 1)$, *IEEE Transactions on Signal Processing* **5**(7): 1772–1779.

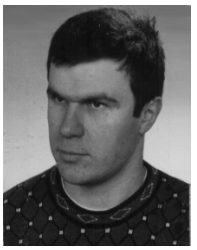
Wang, Z., Jullien, G.A. and Miller, W.C. (2000). An improved residue-to-binary converter, *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications,* **47**(9), pp. 1437–1440.

Wnuk, M. (2008). Remarks on hardware implementation of image processing algorithms, *International Journal of Applied Mathematics and Computer Science* **18**(1): 105–110, DOI: 10.2478/v10006-008-0010-2.

Yassine, H.M. (1992). Hierarchical residue numbering system suitable for VLSI arithmetic architectures, *Proceedings of the IEEE International Symposium on Circuits and Systems, ISCAS '92, San Diego, CA, USA*, pp. 811–814.

Zimmermann, R. (1998). VHDL library of arithmetic units, *Proceedings of the 1st International Forum on Design Languages, FDL'98, Lausanne, Switzerland*, `http://www.iis.ee.ethz.ch/~zimmi /publications/arith_lib_fdl.ps.gz`.

Zimmermann, R. (1999). Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication, *Proceedings of the 14th IEEE Symposium on Computer Arithmetic, Adelaide, Australia*, pp. 158–167.



Tadeusz Tomczak received his M.Sc. and Ph.D. degrees in computer science from the Institute of Computer Engineering, Control and Robotics, Wrocław University of Technology, Poland, in 2002 and 2007, respectively. Currently he is with the same university. His research interests include fast computation hardware, parallel computing including massively parallel processors, and computationally intensive algorithms.

Received: 31 December 2009
Revised: 28 July 2010